MICRO SAINT SHARP

User Guide

for Windows

Version 1.0

Micro Analysis & Design

COPYRIGHT
© 2003 Micro Analysis and Design, Inc. All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, electronic, mechanical, photocopying, manual or otherwise, in whole or in part, without the prior written permission of Micro Analysis & Design, Inc.

TRADEMARKS
Excel, Notepad, Windows, Windows NT, Windows 2000, Windows Me, Windows XP, and Windows Classic are trademarks of Microsoft Corporation.
OptQuest is a trademark of OptTek Systems Inc.
Chart FX is a registered trademark of Software FX, Inc.
.NET is a registered trademark of Microsoft Corporation.
GoLayout Win is a registered trademark of Northwoods.
Sentinel SuperPro is a registered trademark of Rainbow Technologies.

All other products and brand names are trademarks or registered trademarks of their respective owners.

Printed in USA

North America:
Micro Analysis and Design, Inc.
4949 Pearl East Circle, Suite 300
Boulder, CO 80301 USA
Phone: (303) 442-6947
Fax: (303) 442-8274
Email: sales@maad.com
Website: www.maad.com

Europe:
Adept Scientific plc
Amor Way
Letchworth, Herts SG6 1ZA UK
Tel: + 44 (0)1462 480055
Fax: + 44 (0)1462 480213
Web: www.adeptscience.com
email: info@adeptscience.co.uk

Japan:
UDI - Urban Dynamics Institute
Tel: 03(3479)5561
Fax: 03(3479)5559
Web: www.udit.co.jp
email: support@udit.co.jp

Contents

1 • Introduction 1
Micro Saint Sharp Features 2
What's New in This Release 4
About this Manual 5
Document Conventions 7
Using Online Help 7
Displaying Version Information 8
Technical Support 8

2 • Installing Micro Saint Sharp 9
System Requirements 10
Before you Begin Installation 11
Installing Micro Saint Sharp 11
Running the Hardware Key Setup 12
Plugging in the Hardware Key 13
Starting Micro Saint Sharp 14

3 • Understanding Micro Saint Sharp Models 15
Elements of a Micro Saint Sharp Model 16
Data Collection and Analysis 23
How to Create a Model 24

4 • Understanding the Micro Saint Sharp Window 29
Micro Saint Sharp Window Overview 30
Micro Saint Sharp Window Components 31
Window Configurations 39
Manipulating Windows 40
Customizing the Micro Saint Sharp Toolbar 47

5 • Basic Procedures 59
Working with Models 60
Exporting Model Information 63
Searching for Text 65
Replacing Text 66

6 • Network Diagram 69
Creating a Network Diagram 70
Working With Network Objects 70
Editing Text and Objects 80
Printing the Task Network Diagram 86

7 • Model Components 89
Networks 90
Groups 95
Tasks 99
Variables 114
Entity Attributes 122
Watches 129
Functions 131
Scenario Events 139
Snapshots 145
Comments 151

8 • Running the Model 157
Displaying Execution Settings 158
Editing Execution Settings 159
Viewing Trace Data During Simulation 166
Viewing Scheduled Events During Simulation 168
Updating Charts During Simulation 169
Updating Comment Variables During Simulation 170
Viewing Network Animation During Simulation 171
Checking for Syntax Errors 172
Checking for Logic Errors 173
Using the Line Debugger Window 174
Controlling Execution Speed 175

9 • Analyzing Model Data 179
Trace Data Files 180
Task Data Files 180
Queue Data Files 181
Snapshot Data Files 182

Variable Data Files 183

10 • Animator 211
Animator Window Overview 212
Working With Images 218
Charts in the Animator Display 221
Way Points 223
Comments 224
Using Animator Functions 226
Developing an Animator Display 227
Executing the Animator Simulation 228
Copying the Animator Display 228
Printing the Animator Display 229

11 • Expressions. 231
Elements in Expressions 232
Constants 232
Variables 233
Functions 234
Comments 235
Commas 235
Return Statements 236
Arithmetic Operators 237
Assignment Operator 239
Adjustment Operators 239
Logical Operators 240
If-Else Statements 242
While Statements 245
For Loops 246
Do Statements 247
Switch Statements 247
Foreach Statements 249

12 • Built-in Functions 251
Function Syntax and Usage 252
Model Functions 253
Math Functions 265
Distribution Functions 272
Animator Functions 281

13 • Probability Distributions 299
Overview of Probability Distributions 300

References for Probability Distributions 301
Types of Probability Distributions 302

14 • OptQuest 315
How OptQuest Interacts with Micro Saint Sharp 316
Elements of an OptQuest Optimization Model 317
Overview of Using OptQuest 317
Preparing the Model 318
Setting up the Optimization 319
Running the Optimization 329
Interpreting the Results 330
Suggestions for Using OptQuest 331
Factors that Affect Search Performance 332
OptQuest Methodology 339
OptQuest Glossary 344

• Glossary 351

1

Introduction

Micro Saint Sharp is a discrete-event network simulation software package for building models that simulate real-life processes. With Micro Saint Sharp models, you can gain useful information about processes that might be too expensive or time-consuming to test in the real world.

Some common application areas for Micro Saint Sharp include the following:

- Modeling manufacturing processes, such as production lines, to examine resource utilization, efficiency, and cost.
- Modeling transportation systems to examine issues such as scheduling and resource requirements.

- Modeling service systems to optimize procedures, staffing, and other logistical considerations.
- Modeling human operator performance and interaction under changing conditions.

Micro Saint Sharp does not use the terminology or graphic representations of a specific industry. You can model any process as a network of tasks that can be represented in a flow chart diagram. The terms that are used are defined by you.

The degree of complexity in a model is also up to you. You can build a simple, functional model just by drawing a network diagram and filling in the task timing information. You can also build a more complex model that includes dynamically changing variables, probabilistic and tactical branching logic, sorted queues, conditional task execution, image animation during model execution, and extensive data collection.

Whether the model you build is simple or complex, the process of running the model and creating charts from the collected variable data is relatively simple. You simply select the execution settings and the variable data you want collected. Micro Saint Sharp executes the model, using a random number seed you supply to generate task times and routing choices specific to the current run.

Micro Saint Sharp Features

Micro Saint Sharp has the following features:

- A palette of tools for drawing your task network on the screen display.
- Task description windows that you can open from the network diagram or tree view.
- Drop-down menus for defining characteristics such as task time distributions, routing decision types, and queue order.
- Built-in system variables and variables that you can define.

- Built-in functions that you can call from expressions in your model—and the ability to add custom functions.
- Data collection snapshots that you can define to be triggered by clock times or model events.
- Scenario events that you can schedule to occur at specific clock times to change the values of variables or trigger tasks.
- Windows that you can leave open during model execution to display changes in the model state (for example, variable values or scheduled events)
- Symbolic animation of the network diagram during model execution.
- Image animation of the model using your own custom background and images of your choice.
- Collection of snapshot, queue, trace of tasks, and task data.
- Automatic creation of charts from variable data collected during model runs.
- Automatic generation of certain performance measures regarding queues, such as average wait and queue length over time.
- Syntax checker that searches the model for errors in syntax and displays the errors.
- Interoperability so you can exchange data with other software applications.

What's New in This Release

This is the first release of Micro Saint Sharp, but it is built off of the very successful Micro Saint engine. However, it has been completely redesigned to be faster, modular, and more powerful.

Micro Saint Sharp is based upon the robust and fully featured Microsoft C# language. The previous versions of Micro Saint were based upon a much more limiting proprietary language.

This release of Micro Saint Sharp additionally contains the following new features:

- New docking interface allows users to place windows where they are most useful.
- Highly configurable windows that allow you to reposition windows into logical working groups and to temporarily hide windows that you are not using.
- Tree view that displays a list of all model components.
- Task network and Animator palettes that contain the tools you use to construct the task network and Animator diagrams (Animator is an optional add-on component).
- An Output window that displays the trace of execution, including the clock times for events, application errors, whether plugins were loaded successfully, the start and end of simulation, and any syntax errors.
- Enhanced Animator display (previously called ActionView) where you can easily use your own custom images, charts, comments, and way points to the Animator display. Objects can then be dynamically moved around to simulate the model execution.
- A Line Debugger window that increases the resolution for debugging a model simulation.

- A Properties window for all the model items that displays a list of all the item properties. Properties can be changed directly in the Properties window or in the associated dialog box for any model item.
- Multiple charts that can be used to plot variable data as a function of the clock time or as a function of another variable.
- Enhanced functionality on the task network diagram, including Groups (collections of similar tasks that resemble a network and can be minimized) and Comments (the display of text or the values of variables during execution).
- Ability to export all model items in HTML format and all data files in `.xml` format.
- Fully customizable toolbar.

About this Manual

The *Micro Saint Sharp User Guide* contains the following chapters and glossary:

Chapter 1 Introduction
This chapter contains an overview of Micro Saint Sharp, lists the contents of the guide, explains how to use the online help feature, and outlines the conventions used in documentation.

Chapter 2 Installing Micro Saint Sharp
This chapter describes system requirements and how to install and start Micro Saint Sharp.

Chapter 3 Understanding Micro Saint Sharp Models
This chapter describes the various elements of a Micro Saint Sharp model. Sections include the task network, entities, variables, functions, data collection snapshots, scenario events, data collection, and analysis. The chapter concludes with a general discussion of the steps involved in creating, debugging, and running a Micro Saint Sharp model.

Chapter 4 Understanding the Micro Saint Sharp Window
This chapter presents an overview of the components of the Micro Saint Sharp window and how to move, resize, hide, display, close, and dock the window panes. How to customize the toolbars is also described.

Chapter 5 Basic Procedures
This chapter describes how to create, open, and save models, export network components, and search and replace text.

Chapter 6 Network Diagram
This chapter describes how to create a network diagram.

Chapter 7 Model Components
This chapter describes the parameters for model components: tasks, networks, comments, groups, variables, entity attributes, functions, events, and snapshots.

Chapter 8 Running the Model
This chapter describes the settings that control model execution, including speed options. How to check a model for syntax and logic errors is also discussed.

Chapter 9 Analyzing Model Data
This chapter describes methods used to analyze the model data and how to create charts displaying the values of variables during execution. How to export and open trace, queue, snapshot and task data files is also discussed.

Chapter 10 Animator
This chapter describes how to build image animation into your model. How to add a background, images, comments, way points, and charts is described. Animator is an optional add-on component.

Chapter 11 Expressions
This chapter explains the constants, variables, functions, comments, operators, statements, and loops used to create Micro Saint Sharp expressions.

Chapter 12 Built-in Functions
This chapter discusses built-in Micro Saint Sharp functions, including Model, Math, Distribution, and Animator functions.

Chapter 13 Probability Distributions
This chapter discusses the twenty probability distributions that are available in Micro Saint Sharp.

Chapter 14 OptQuest
This chapter describes the OptQuest feature, which is automatically included with Micro Saint Sharp. OptQuest enhances Micro Saint Sharp by automatically searching for and finding optimal or near-optimal solutions to simulation models.

Glossary
The glossary contains a listing of the terms used in Micro Saint Sharp.

Document Conventions

Micro Saint Sharp documentation uses the following terms and typographical conventions:

- File names, variables, and program code are shown in this `typeface`.
- Labeled buttons, menu commands, and menu options are in bold. For example, Click **OK** to close the dialog box.
- Information that the user needs to enter is in Courier New font. For example, enter `BO90001` in the text box.

Using Online Help

Micro Saint Sharp contains extensive online Help to assist you as you work. To display help contents, select **Help** from the **Help** menu.

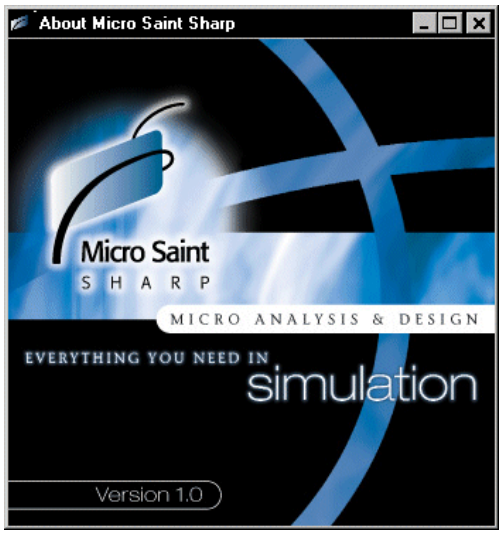
Displaying Version Information

You can view version information for Micro Saint Sharp. You may want to know the version number when you are requesting technical support.

To display version information:

1. From the **Help** menu, select **About**.

   The About Micro Saint Sharp dialog box information displays. The version number of the software is indicated.



2. To close the dialog box, click the X in the upper right corner.

Technical Support

You can e-mail any questions you have regarding Micro Saint Sharp to the following address:

techsupport@maad.com

2

Installing Micro Saint Sharp

This chapter presents the instructions for installing Micro Saint Sharp. In this chapter, you will learn the following:

- What you need to know—and what you should have on hand—before you start the installation
- What the system requirements are
- How to install the software for the hardware key
- How to plug in the hardware key
- How to install Micro Saint Sharp
- How to start Micro Saint Sharp

System Requirements

The minimum system requirements to run Micro Saint Sharp are the following:

- 90-Megahertz Intel Pentium-class processor.
- Microsoft Windows Server 2003, Windows XP Professional, Windows XP Home Edition, Windows 2000, Windows Millennium Edition (Windows Me), and Windows 98.

  Operating systems must support the .NET Framework 1.1. The .NET Framework 1.1 Redistributable cannot be installed on 64-bit computers; Windows NT 4.0 Terminal Server is not supported.
- Microsoft Windows NTR 4.0 Service Pack 6a.
- 32 megabytes (MB) of RAM, 96 MB recommended.
- 150 MB free hard disk space.
- 800 x 600 or higher-resolution display with 256 colors.
- Microsoft mouse or compatible pointing device.
- CD-ROM.

Before you Begin Installation

Before you install Micro Saint Sharp, check that you have the following items:

- Micro Saint Sharp Program CD
- Micro Saint Sharp User Guide
- Hardware Key

Installing Micro Saint Sharp

To install Micro Saint Sharp:

1. Insert the Micro Saint Sharp CD into your CD-ROM drive.

   If the CD fails to start the Setup process, then follow these steps:

   a. Open Windows explorer.
   b. Navigate to the Micro Saint Sharp CD-ROM and double-click `Micro Saint Sharp Gold Msi` or `Micro Saint Sharp Silver Msi`.

      The file on the CD-ROM is based upon which version of Micro Saint Sharp you purchased.

2. Follow the instructions that display on the screen to complete the installation process.

   You may be prompted to install .NET Framework. Micro Saint Sharp uses the .NET Framework platform; version 1.1 is required. For details about .NET Framework, see www.microsoft.com/net.

The installation program copies all of the Micro Saint Sharp software, help files, and sample model files into the `Program Files\Maad\Micro Saint Sharp` directory or into the directory you specify. Micro Saint Sharp also places an icon on the desktop from which you can launch Micro Saint Sharp.

Note:
Micro Saint Sharp additionally includes sample models and tutorials on the installation CD. These are automatically installed.

Running the Hardware Key Setup

To run Micro Saint Sharp, you need to plug a special hardware key into you computer. For your computer to recognize the hardware key, you need to install the drivers first.

To install the drivers for the hardware key:

1. Navigate to the location where you installed Micro Saint Sharp.

   The default location is `c:\Program files\Maad\Micro Saint Sharp`.

2. Open the `Rainbow Setup` folder.

3. Double-click `Rainbow Setup.exe`.

   Follow the instructions that display. Once you have successfully installed the drivers, you can plug in the hardware key.

Plugging in the Hardware Key

Once you have installed the drivers, you can plug the hardware key into the parallel or USB port of your computer. This key, labeled Sentinel SuperPro™, is included when you purchased Micro Saint Sharp.

If you do not install a hardware key, you can only run a model and cannot save it.

To plug in the hardware key:

1. Plug the hardware key into the parallel or USB port on your computer.

   If you are not sure which port is the parallel or the USB port, consult the manual for your computer.

   If you need to use your parallel port for a connection to another parallel device such as your printer, connect the cable onto the back of the hardware key. The hardware key will be between the parallel port and the cable connector for the parallel device. You must turn the parallel device (such as your printer) on for Micro Saint Sharp to recognize that the key is present.

Starting Micro Saint Sharp

To start Micro Saint Sharp:

1. Click the **Start** button, and then point to **Programs**.

2. Select **Micro Saint Sharp** and then select the Micro Saint Sharp icon and program name. You can alternatively double-click the Micro Saint Sharp icon on the desktop.

   Micro Saint Sharp opens.

3

Understanding Micro Saint Sharp Models

This chapter presents an overview of Micro Saint Sharp models. Sections discuss the components of a Micro Saint Sharp model, including the task network, variables, functions, entity attributes, data collection snapshots, scenario events, data collection, and data analysis. The chapter concludes with a general discussion of the steps involved in creating, debugging, and running a Micro Saint Sharp model.

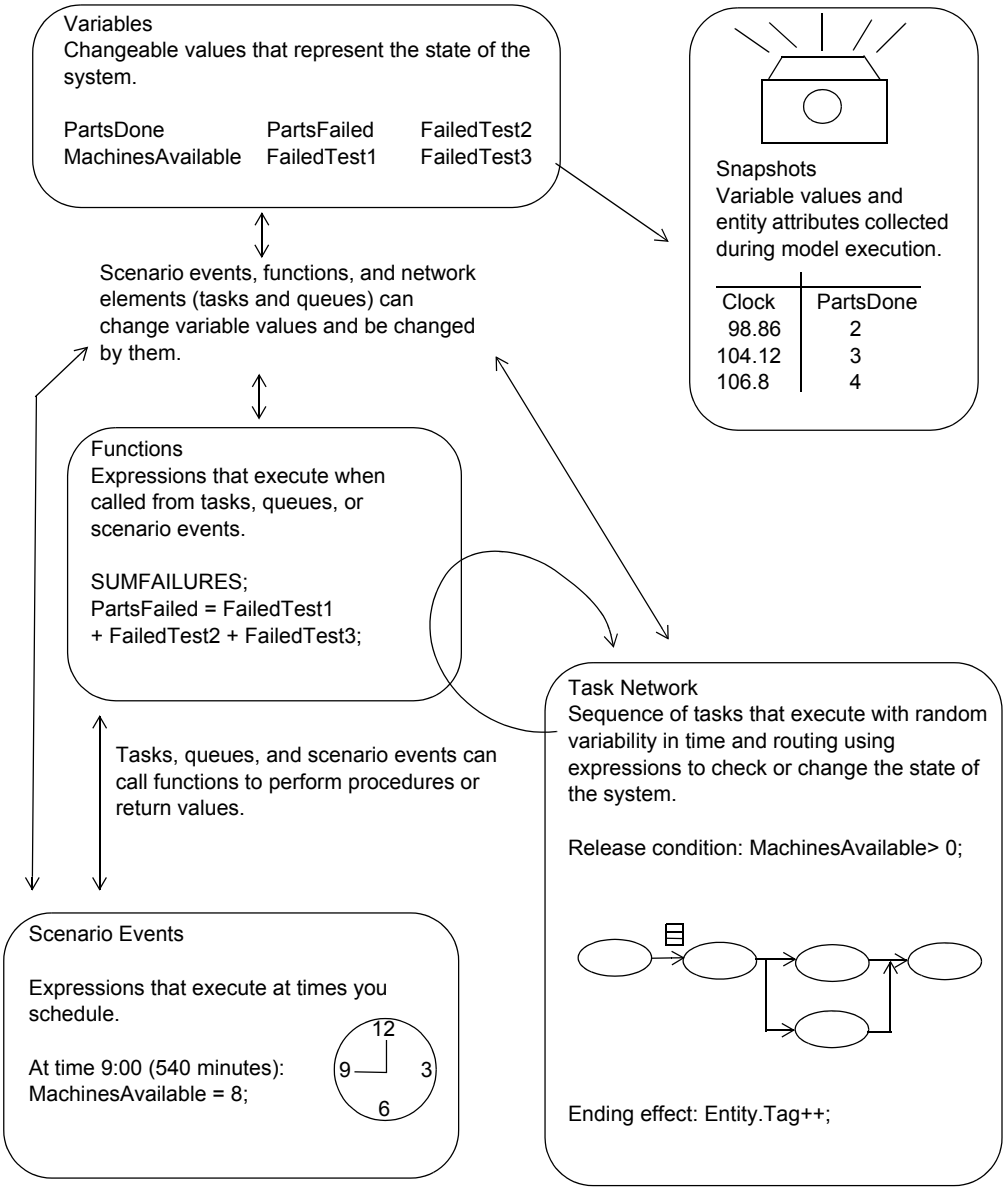
Elements of a Micro Saint Sharp Model

A Micro Saint Sharp model consists of the following elements:

- A task network that executes a sequence of tasks with variability under conditions and in sequences that you specify.
- Entities that are the conceptual objects that travel through a task network.
- Variables you define to represent changeable system states or characteristics, with values that change as effects of tasks, queues, or scenario events.
- Functions that return values or perform procedures when they are called in tasks, queues, or scenario events.
- Snapshots that collect the values of particular variables when triggered by conditions that you specify.
- Scenario events that you schedule to occur at specific clock times to change the values of variables.

These elements and their interrelationships are shown in the following diagram. The diagram does not show a logical flow of model execution, but illustrates the general relationships between the aspects of a model.

Elements of a Micro Saint Sharp Model



Task Network

The minimal elements required for a Micro Saint Sharp network are tasks and the paths connecting the tasks. A network can also include decision nodes when more than one path leaves a task or subnetwork.

Entities are the conceptual objects that travel through a task network. The entity may represent a physical object, such as a part being built on a production line. It may also represent a person performing the tasks. Or, the entity may simply trace the sequence of tasks and not correspond to anything physical. Default attributes are associated with each entity and track the state of the entity. You can add other attributes to an entity.

Tasks, Paths, and Decisions

When you execute a model, an entity starts at the task you designate as the starting point. The entity then travels along the paths, executing each task it encounters. When multiple paths leave a task, the decision type determines the paths the entity should take. Multiple entities are created when an entity splits to follow more than one path out of a task.

The execution time for each task varies within the parameters you supply (usually distribution type, mean time, and standard deviation). The sequence of tasks that the entity follows can also vary if you include probabilistic or tactical routing decisions. In a probabilistic decision, you specify the probability of each following task and Micro Saint Sharp selects a task randomly within these probabilities. In a tactical decision, you specify the system conditions under which each following task can be selected. You can define these conditions and probabilities using expressions, operators, and variables that represent the state of the system.

Additionally, as each task executes it can alter the state of the system with expressions called “effects.” Effects are executed either at the beginning or at the end of a task. For example, the beginning effect of a machine task might decrease the number of available machines by one while the ending effect would increase the number by one.

Each task can also check the state of the system before it executes and delay execution until a certain condition, called a release condition, is met. For example, assume you define a variable named `Workers` that tracks the number of available operators. If you have a task that requires an operator to perform it, the condition `Workers > 0` must be true for the task to execute.

Queues

Usually when you define a release condition for a task, you also place a queue in front of the task as a waiting area. When the model is running, entities move into the queue in front of the task when the task is busy. The entities wait in the queue until the release condition is true. Then they are released in the order you specified when you define the queue.

You can specify that the entities be ordered within the queue as First In First Out (FIFO), Last In First Out (LIFO), or Sorted (where you supply the priority expression for sorting).

Networks

When a task in a network is very complex, you can divide it into more specific tasks; this is also called a hierarchical decomposition. You do this by defining the task as a network—or subnetwork, because it is a lower-level network than the main task network. The subnetwork can contain all the same elements as the main network—tasks, paths, decisions, queues, and subnetworks at a still lower level.

When an entity traveling through the model is released into a subnetwork, it starts at the first task in the subnetwork and proceeds through the subnetwork in the same way that it would proceed through the main network. Multiple entry and exits are allowed into each network.

Entities

Entities are the conceptual objects that travel through a task network and are used to track the objects that move through the system. Default entity attributes include the ones in the following list. You can also create your own entity attributes.

- `Entity.Duration` records the time each entity spends in the current task or queue.
- `Entity.Group` records the group identity of the entity being processed.
- `Entity.ID` records the task number in which the entity is in.
- `Entity.Tag` records the identity of each entity when you have multiple entities traveling through a network.
- `Entity.Time` tracks the closest scheduled event time for the entity.
- `Entity.Type` records the event type for which entity is scheduled. Event types can be one-time or repeating.
- `Entity.UniqueID` maintains the unique ID for the entity as defined by Micro Saint Sharp.

Variables

Variables are an important element in any model because they keep track of the state of the system. Five system variables are automatically created for each Micro Saint Sharp model and you can additionally define variables specific to each model.

The variables you define can keep track of whatever is appropriate for the model—how many items are processed, what the current temperature is, or whether a machine is On or Off. Variables give different tasks, queues, and scenario events a way to interact with each other because they can evaluate and modify variable values.

Default system variables include the following:

- `Clock` records elapsed time (in simulation time units) since the beginning of model execution.
- `Distributions` consists of all the model distributions. Distributions are used to determine task execution times.
- `Entity` consists of all the entity variables and functions.
- `Model` consists of all the model functions.
- `Animator` consists of all the Animator components (Animator is an optional add-on component).

Functions

Micro Saint Sharp uses two types of functions: built-in functions, which are available to all models, and custom functions, which you define for use within a particular model. Both function types work in a similar manner. You can include the function's name or call the function in any expression. When Micro Saint Sharp encounters the function, it executes the function and returns a value that can be used in the expression.

Built-in functions include the following categories:

- Modeling functions, which perform actions such as starting or stopping tasks, and pausing or stopping model executions.
- Mathematical functions, which perform mathematical operations, such as minimum, maximum, and trigonometric functions.
- Distribution functions, which generate random numbers for task execution according to a certain distribution.
- Animator functions, which create, dispose, and move Animator images and control the attributes of the Animator display (Animator is an optional add-on component).

Custom functions are particularly useful for calculations or procedures that you want to execute at more than one place in a model. For example, you might create a function to calculate machine utilization and use it to calculate utilization for various machines.

When functions are called in Micro Saint Sharp, they are followed by parentheses, such as `CalculateTime( )`. To return a value from a function, the function itself must be defined with a return value (for example, an "integer").

Scenario Events

Scenario events provide a way for you to cause events to occur at specific times during model execution. These can be one-time events, or events that repeat at regular intervals. Scenario events are often used to change variable values, thereby changing the state of the model. For example, you might have a variable called `temperature` that would increase at 15-minute intervals during the day and decrease during the night. You could then make the times required for tasks such as warming up an engine be contingent upon the current temperature.

Snapshots

Snapshots provide a way to collect values of variables at specified times during model execution. You can specify snapshots to be gathered at specific clock times (one-time or repeating), when a task begins or ends, when an entity enters or leaves a queue, or when a model run ends.

Data Collection and Analysis

Micro Saint Sharp stores a wide variety of model information during model simulation in four different data files: trace data, task information, queue information, and snapshots. For all of these except the trace data, you enable the collection of data in the execution settings for the model.

- Trace data files record the clock times for beginning effects, ending effects, and scenario events. Application errors, whether plugins were loaded successfully, and the start and end of a simulation are also included. Trace data displays in the Output window as a model executes.

- Task information files record the task number, task name, minimum task time, maximum task time, mean task time and standard deviation, and number of times a task was performed.

- Queue information files record the queue ID, queue name, and number of times an entity entered the queue. Also recorded are the minimum, maximum, and mean time/standard deviation that the entity spent in the queue and the minimum, maximum, and mean time/standard deviation that the entity spent waiting in the queue before being allowed to execute the task.

- Snapshot data files contain the values of selected variables at specified times during model execution.

You can save trace data file to a `.txt` file for additional analysis. You can export task information, queue information, and snapshot data files from Micro Saint Sharp in `.xml` format and open them in another application such as Excel for analysis. You can also save snapshot files as `.res` files, which are tab delimited ASCII text files.

You can additionally use the Chart feature in Micro Saint Sharp to chart the values of variables as a function of the clock value or any other variable as the model is executing. Micro Saint Sharp provides twenty different chart types that you can use for data analysis. You can format print, copy, and paste charts, and save charts as an image file.

How to Create a Model

This section presents an overview of the modeling process and provides some tips and pertinent questions to help make sure you are on the right track at each step. It also emphasizes the early steps to perform before you begin building the model in Micro Saint Sharp. If you work through the tutorials provided on the Micro Saint Sharp CD, you will perform most of the standard model-building steps described here.

While the steps in this list are numbered, keep in mind that the modeling process is iterative. You may go back and forth between various steps in the process as new ideas occur to you, and you may go through several revisions of the model before you reach a final version. There is no one correct sequence that you need to follow.

In general, however, you develop a model through the following process:

1. **Determine the questions you want to answer.**

   Before you start building a model, be sure to think about what it is you are trying to determine. If you try to model the entire manufacturing process at your plant when all you really need to do is determine how many of a certain type of machine you need, you will expend a lot of extra effort. Build the model to answer your questions, rather than thinking of the questions after you build it.

2. **Analyze the process you want to model.**

   Before you can draw a diagram of the process, you need to understand the process thoroughly. Determine what tasks are involved in the process, the sequence in which they are performed, what resources they use, whether there are any restrictions on when they can be performed, and how they affect the overall system being modeled.

As you determine what tasks are involved, you also need to determine how long each task requires to execute and how the times for each task are distributed. You need to determine the probability distribution that characterizes the task, and what values are appropriate for the distribution parameters (usually mean and standard deviation). You may need to gather data to determine these values, or at least obtain estimates from subject matter experts.

You can use various statistical curve fitting packages to determine the appropriate probability distribution for a task. These packages import data from an Excel spreadsheet, fit the data to a curve, and provide the necessary parameters (such as mean or standard deviation) for Micro Saint Sharp. The packages are available from several software distributors; call MA&D for further information.

3. Draw the network diagram.

Once you have analyzed the process, drawing the task network diagram in Micro Saint Sharp is easy. Just drag the network diagram objects from the Palette tab to the positions where you want them. Then draw the paths from each task to any tasks or network that can follow that task.

As you are drawing the diagram, remember that you can use tasks to represent mental processes, physical processes, and processes that are not actually performed. For example, you can use a task to represent the decision making process that can result in several possible alternative paths to following tasks. Or, you may use a "spinner task" to randomly generate arriving entities, such as people or parts. A spinner task is a task with a path that goes back to itself. The tutorials and sample models included on the Micro Saint Sharp CD provide examples of spinner tasks and other techniques for simulating different types of situations.

4. **Define how the jobs, decisions, and queues operate.**

   This is where you use the information you gathered when you analyzed the process in step 2. Double-clicking a network object on the network diagram opens a Task Description dialog box where you can define timing information, execution constraints, queue information, decision routing, and the effects of the task on the system.

   In defining conditions that check the state of the system and effects that change the state of the system, you need to create expressions that use variables. You may also want to call some custom functions to execute expressions that you use in several places. Whether you define the variables and functions when you first use them or later on is up to you.

5. **Define the variables and system changes.**

   If you finish defining the elements in the network diagram before you start defining the variables, you already have most of the information that you need for the variables. In addition to these variables, consider whether there are any other system characteristics you should represent with variables. Have you represented all environmental changes in the system that might affect task performance? Have you included all important counts and measurements that change as the simulation progresses? Have you included all the quantities that you want to use as input to or output from the system?

   As you are defining variables, consider whether you have accounted for all changes in each variable's value as task or queue effects. If any of the variables you define change as a result of time passing rather than as effects of task or queue activity, you need to display the Event Queue and define scenario events for these changes.

6. Define any custom functions called in the jobs, queues, or scenario events.

Custom functions are useful when you need to perform the same calculation or procedure several times in a model. Instead of retyping the necessary expressions every place you need them, you can first create the function once for a model. Then, wherever you need the calculation or procedure to be performed in the model, you simply type the name of the function followed by parentheses ( ).

7. Check for errors.

The built-in error checker automatically searches the model for errors in syntax. Any detected errors display in the Output window and a brief explanation of each error is provided. Double-clicking an error displays the dialog box where the error occurs where you can correct it.

8. Run and debug the model.

After you start the model execution menu, the symbolic animation of the network diagram displays the entities as they travel through the system. Micro Saint Sharp has several different built-in tools that you can use to help you locate any logic errors in the model.

- The Watches window displays the values of selected variables during model execution.
- The Event Queue window displays the values of scheduled events.
- The Output window displays the actions that occur when you run the model along with any syntax errors.
- The Line Debugger window increases the resolution for debugging a model simulation. When you step through model simulation, execution occurs one line of code at a time.
- Comment boxes can display the values of variables and expressions at any location in the task network.

Additionally, the snapshot data, queue information, and task information data files contain valuable information that you can use to help debug the model.

9. Add data collection.

Once the model is running smoothly, you can define snapshots to collect the values of variables at specified points during model execution and run the model once or multiple times with snapshots enabled. The values you collect can provide further validation that the model is operating correctly or can alert you to possible problems. You can also set the collection of queue and task information data.

10. Analyze the data.

You can export the trace of task, task information, queue information, and snapshot data for analysis. Additionally, you can use the Chart tool in Micro Saint Sharp to generate twenty different types of charts for variable data. As you analyze the data, you will start to answer the questions you originally asked—and you may discover more questions that lead you to further development of the model.

4

Understanding the Micro Saint Sharp Window

Micro Saint Sharp uses a modified form of the MDI (Multiple Document Interface) made available by Microsoft .NET Framework. MDI provides a parent container window that contains several other different window panes. This configuration allows you to reposition windows into logical working groups and to temporarily hide windows that you are not using. The Micro Saint Sharp window is highly configurable, and changes to the configuration are saved when you save a model.

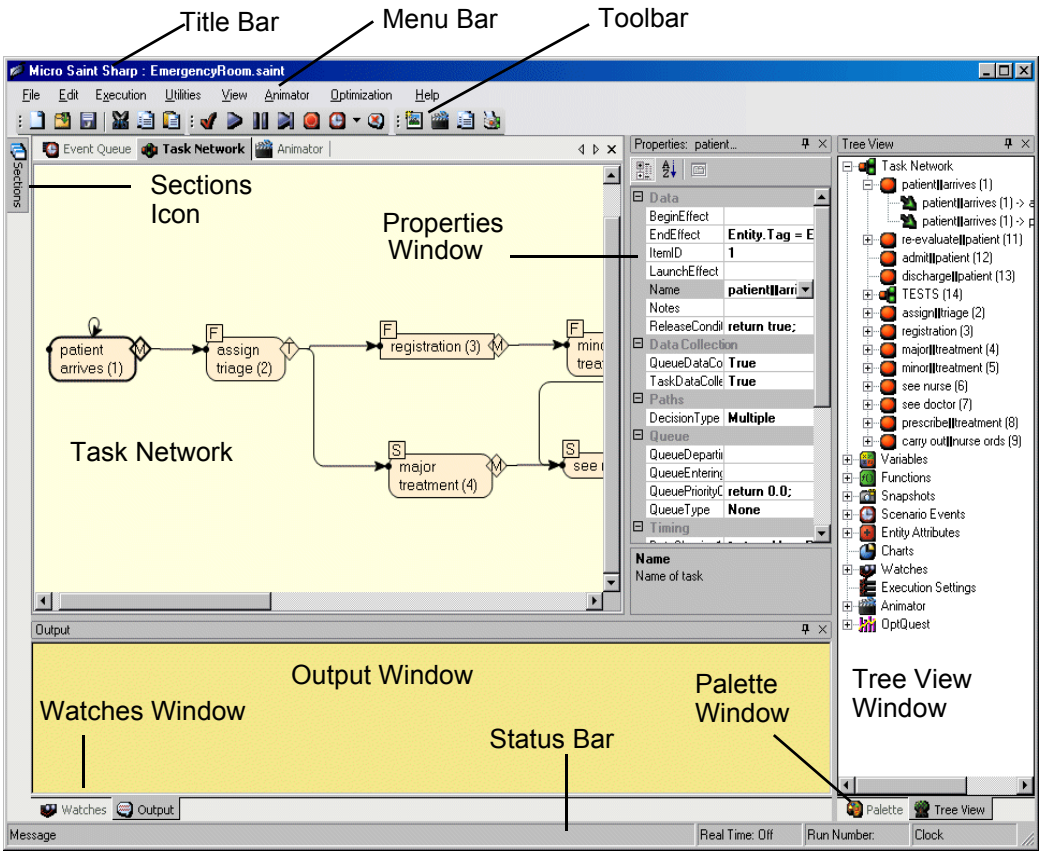
Micro Saint Sharp Window Overview

The default configuration for the Micro Saint Sharp window consists of a container window with a title bar, menu bar, toolbar, and status bar. Within this container, the following elements exist as separate window panes: Sections (displays as an icon in the default configuration), Event Queue, Task Network, Properties Window, Tree View, Palette, Watches and Output.

You can move, resize, hide, and dock all of the windows. You may want to use different configurations during different stages of model development. The screen layout is saved with the model.

Note:
You can restore the default window configuration at any time by selecting **Layout** from the **View** menu and then selecting **Reset Layout**.

The following illustration shows the components of the Micro Saint Sharp window. Each of these components are described in greater detail in the remaining sections of this chapter.



Micro Saint Sharp Window Components

This section describes the window components in the main Micro Saint Sharp window. For descriptions of the Animator window, see “Animator Window Overview” on page 212.

For details on resizing, moving, hiding, and docking the windows, see “Manipulating Windows” on page 40.

Title Bar

The title bar displays at the very top of the Micro Saint Sharp window and contains the name of the currently open model. Micro Saint Sharp models have the extension `.saint`.

Menu Bar

The menu bar displays below the title bar at the top of the Micro Saint Sharp window and contains the File, Edit, Execution, Utilities, View, and Help menus.

- The File menu includes basic model commands such as New Model, Open Model, and Save Model.
- The Edit menu includes basic edit commands such as Copy, Cut, Paste, and commands for adding elements to the task network diagram, such as Add Task, Add Comment, and Add Network.

> ✓ Note:
> Most of the Edit commands are also available when you right-click the mouse either in the task network diagram or tree view.

- The Execution menu includes commands controlling execution, such as Begin, Step, and Halt Simulation.
- The Utilities menu contains commands to display trace data, charts, and the event queue, and to enable animation. Commands for updating comment variables, controlling execution speed, grouping and ungrouping objects, aligning objects, and viewing the model in HTML format are also included.
- The View menu contains commands to zoom the display, display different windows, reset the screen layout, and reset the toolbars to the default configuration.

- The Optimization menu contains commands to start and Halt an optimization.
- The Help menu contains commands to display version information and to open online help.

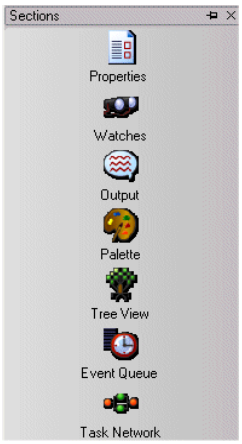
Toolbar

The Micro Saint Sharp toolbar displays below the menu bar and provides access to commonly used commands. By default, these include the following: New Model, Open Model, Save Model, Cut, Copy, Paste, Check for Errors, Begin Simulation, Pause Simulation, Step Simulation, Halt Simulation, Simulation Speed, and Clear Output.

When you move the cursor over each button, flyover text displays the command and the associated shortcut key if available.

You can customize the toolbar by moving the position of toolbar sections, displaying or removing toolbar sections, adding and removing default buttons, adding new toolbar sections, controlling whether buttons display as image, text, or image and text, and adding additional buttons for commands to any of the sections. For details, see “Customizing the Micro Saint Sharp Toolbar” on page 47.

Sections Window

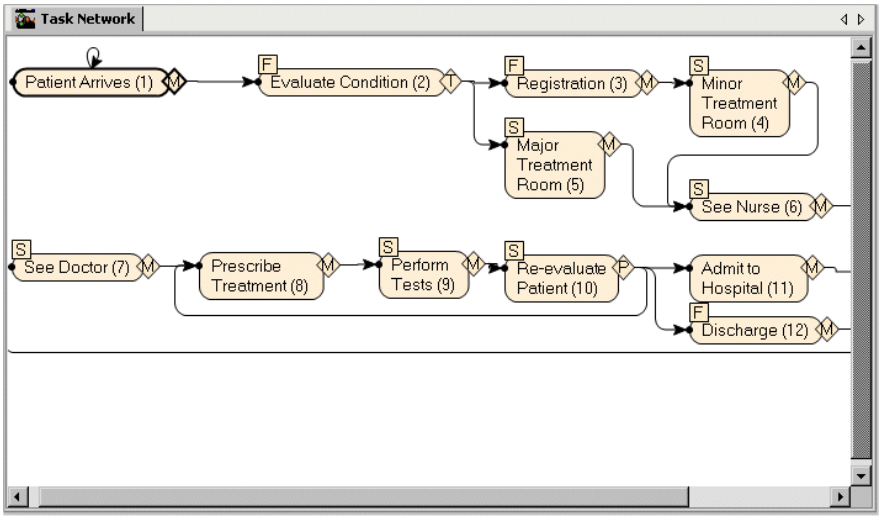


The Sections window contains a list of the windows that you can display in Micro Saint Sharp and is intended to be a shortcut to quickly open a window. Windows include the following: Event Queue, Line Debugger, Output, Palette, Properties, Task Network, Tree View, and Watches.

In the default configuration, the Sections window pane displays as a tab docked on the left side of the Micro Saint Sharp window. To display the Sections window, just move the cursor over the Sections tab and the full window displays. You can then select the window that you want to display.

Task Network Window

The Task Network window displays the graphical depiction of the task network. This is the window where you construct the task network diagram and view the model execution.



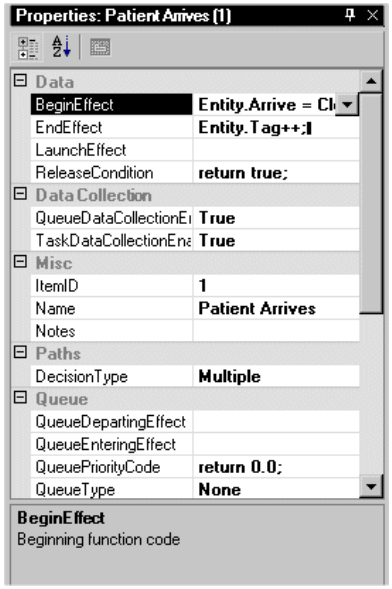
Event Queue Window

The Event Queue window displays the list of events as they occur during model execution. These include the execution of tasks and scheduled events. You create the scheduled events for the simulation. Scheduled events can be one-time events or they can repeat at regular intervals. The group, ID, tag, and time, and type of event displays.

Beginning effects are highlighted in navy, ending effects are in green, suspended beginning effects are in blue, and suspended ending effects are in lime. Evaluate queue is in fuchsia, leaving queue is in purple, and waiting without queue is in red. And finally, scenario events are in yellow and snapshots are in maroon.

Event	Group	ID	Tag	Time	Type
EndingEffect	0	assign	1	4.43622446	Task
EndingEffect	0	patient	1	24.5122013	Task
ScenarioEvent	0	ScenarioEvent10	0	480	ScenarioEvent
ScenarioEvent	0	ScenarioEvent11	0	840	ScenarioEvent

Properties Window



The Properties window displays the properties of the model item selected in the task network diagram, such as network, task, or group. It also displays the properties of items that you select in the tree view such as variables, functions, scenario events, charts, and snapshots. The properties that display in the Properties window also display in the dialog box for the associated item.

The Properties window functions like any folder list. To expand an item, click the adjacent plus sign. To collapse an item, click the adjacent minus sign.

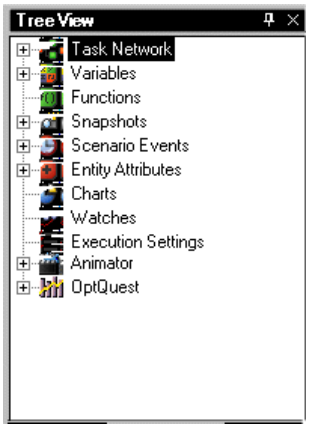
Some of the fields in the Properties window have an associated drop-down list box. To display the list box, click in the text box and a down-facing arrow displays. Click the arrow and a selection list appears.

A detailed description of any selected item automatically displays at the bottom of the Properties window.

To sort the items in the Properties window, do one of the following:

- To sort the properties alphabetically, click the **Alphabetic** button .
- To sort the properties by category, click the **Categorize** button .

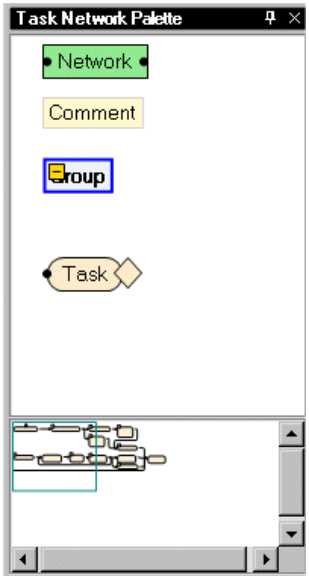
Tree View Window



The Tree View window displays a hierarchical list of the model components, including the tasks, decision nodes, comments, groups, and subnetworks in the task network, and the variables, functions, snapshots, scenario events, entity attributes, charts, watches, execution settings, Animator components, and OptQuest components.

To expand a component in the tree view, click the plus sign adjacent to the item. To collapse a component, click the minus sign adjacent to the item.

Task Network Palette

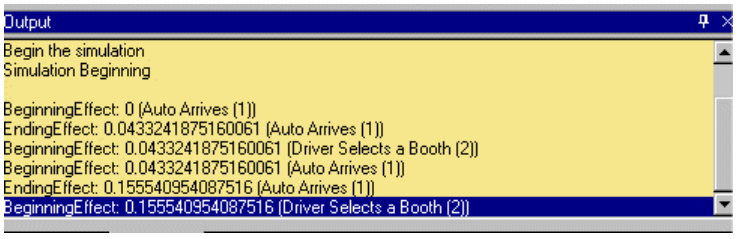


The Task Network palette displays the tools you use to construct the network diagram. These include tools for adding networks, comments, groups, and tasks. To add one of these items to the network diagram, you click the item and drag it onto the network. You can then open a Description dialog box for the item and enter the defining parameters.

The relative position of the active display in the network diagram displays at the bottom of the palette and is outlined in blue-green.

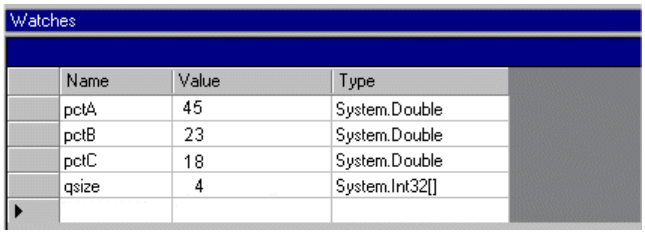
Output Window

The Output window displays the trace of the execution, which consists of the actions that occur when you run the model. The clock times for beginning effects, ending effects, and scenario events are listed. Application errors, whether plugins were loaded successfully, and the start and end of a simulation are also included. Any syntax errors are indicated and you can use them for debugging purposes. You can save the information in the Output window to a file or to the Windows clipboard and clear the information in the window before running a new simulation.

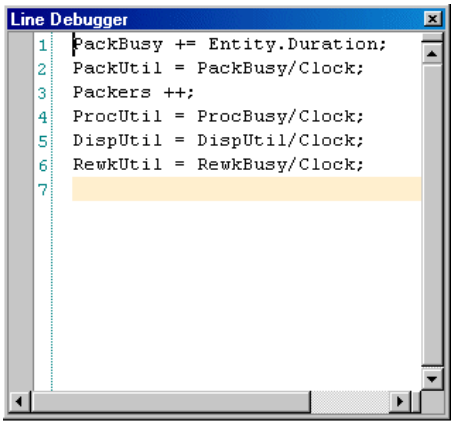

Output
Begin the simulation
Simulation Beginning
BeginningEffect: 0 (Auto Arrives (1))
EndingEffect: 0.0433241875160061 (Auto Arrives (1))
BeginningEffect: 0.0433241875160061 (Driver Selects a Booth (2))
BeginningEffect: 0.0433241875160061 (Auto Arrives (1))
EndingEffect: 0.155540954087516 (Auto Arrives (1))
BeginningEffect: 0.155540954087516 (Driver Selects a Booth (2))

Watches Window

The Watches window displays the values of variables during model execution. You select the variables that you want to display by creating a watch for the variable. The variable name, value, and type are displayed.



Line Debugger Window



The Line Debugger window is a separate window that you can use to increase the resolution of execution so that you can debug a model simulation. When the Line Debugger window is open and you step through the model simulation, execution occurs one line of code at a time. When the Line Debugger window is not open and you step through simulation, execution occurs one effect at a time rather than one line of code at a time. You can only enable line debugging between model runs.

The Line Debugger window does not display in the default configuration. You can display it from the Sections window or by selecting Line Degugger from the Windows menu. You must also enable the display of the Line Debugger in the Execution settings (see page 161).

Status Bar

The status bar displays at the bottom of the Micro Saint Sharp window. Any execution-related messages display on the left side of the status bar. The clock time, the run number, and a flag indicating whether the model is being run in realtime mode (see page 161) display on the right side of the status bar.

The status bar also displays the cursor coordinates in the Animator window when you click in the window (Animator is an optional add-on component).

Window Configurations

Micro Saint Sharp provides two default window configurations or views, the Edit and the Execution configurations. These views were configured with the optimum windows for each in each mode.

- The Edit configuration displays the Task Network, Properties, Tree View, Palette, Watches, and Output window.
- The Execution configuration displays the Task Network, Watches, and Output window.

To use the Edit configuration:

From the **View** menu, select **Layout**, and then select **Edit**.

A check mark displays adjacent to the Edit label to show it is selected.

To use the Execution configuration:

From the **View** menu, select **Layout**, and then select **Execution**.

A check mark displays adjacent to the Execution label to show it is selected.

You can set the change between Edit and Execution mode to occur automatically when you run the model using the Enable Layout Change setting in the Execution Settings Properties window. For details, see page 163.

Manipulating Windows

Micro Saint Sharp uses a docking manager interface made available by Microsoft .NET Framework. This type of interface provides a parent container window for two types of windows, Document and Tool windows. These windows share some properties but also have unique properties of their own.

- Document windows are dynamically created when you open or create files or other items. Document windows consist of the Task Description window and the associated dialog boxes for a selected variable, function, snapshot, entity attribute, and chart.

  Document windows can be docked and can be displayed in a vertical or horizontal tabbed format.

- Tool windows are defined by Micro Saint Sharp and consist of the Task Network, Event Queue, Sections, Tree View, Property, Output, Watches, Tool Palette, and Line Debugging windows.

  Tool windows can be floating, can be docked, hidden, displayed in a vertical or horizontal tabbed format, and have the auto-hide property.

Floating Windows

When a window is floating, you can move it around as a separate window that floats on top of the other windows. You can move a floating window outside of the main Micro Saint Sharp window. Only Tool windows can be floating.

To float a window:

1. Right-click the title of the window, such as Tree View or Properties.

2. Select **Floating** from the menu that displays so that a check mark appears adjacent to the Floating item.

   The window appears as a separate window at the upper left corner of the screen.

Making Windows Dockable

When a window is dockable, you can move it with the mouse so that it snaps to the closest side of the Micro Saint Sharp window. You can make both Document and Tool windows dockable, but they show different behaviors.

To make a window dockable:

1. Right-click the title of the window, such as Tree View or Properties.

2. Select **Dockable** from the menu that displays so that a check mark displays adjacent to the Dockable item.

   Document windows automatically appear as separate windows so you can move them to a docked location. Tool windows remain in the same location until you move them.

Docking Windows

Once you set a window to be dockable, you can dock it to any of the sides of the main Micro Saint Sharp window. You can dock both Document and Tool windows.

To dock a window:

1. Drag the window toward an edge of the Micro Saint Sharp window until you see a superimposed outline in the location you want.

   The outline of the object you are dragging indicates the orientation it takes when released. The position of the mouse pointer determines where the window is placed, not the edges of the window.

2. Release the mouse button.

✓ Note:
To move a dockable window without snapping it into place, press CTRL while dragging it to the location.

Displaying Tab Groups

You can display tabbed Document and Tool windows adjacent to each other so that they can be viewed at the same time. Both Document and Tool windows can be displayed as tab groups. Tool windows can only be tabbed if they are not selected to be dockable.

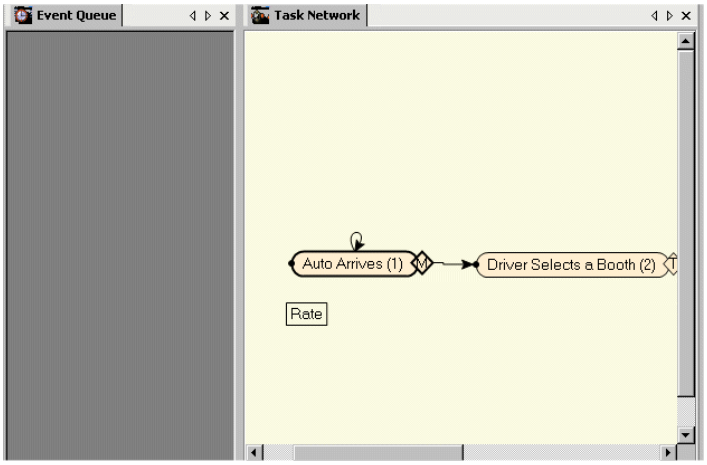
To display a tab group:

1. Right-click the title of any of the tabs in the Document window.

2. Do one of the following:
   - To display the tab in a horizontal format, select **New Horizontal Tab Group** from the menu that displays.
   - To display the tab in a vertical format, select **New Vertical Tab Group** from the menu that displays.

   The window rearranges to the selected format.

3. Repeat the previous step for any of the other windows.



✓ Note**:**
To restore the previous orientation, right-click the title of the window, and select **Move to Previous Tab Group**.

Hiding Windows

Hiding a window completely removes the window from the display and frees up more working space. You can only hide Tool windows. Hiding a window is the same as closing the window.

To hide a window:

1. Right-click the title of the window, such as Tree View or Properties.
2. Select **Hide** from the menu that displays. You can alternatively click the ✕ in the upper right corner of the window.

   The window is removed from the display.

Redisplaying Hidden Windows

Once a window is hidden, you can redisplay it using the View menu.

To redisplay a hidden window:

1. From the **View** menu, select **Windows**.
2. Select the window.

   The window redisplays in the last position it occupied.

   You can also redisplay a window using the Sections tab. For details, see “Sections Window” on page 34.

Auto Hiding Windows

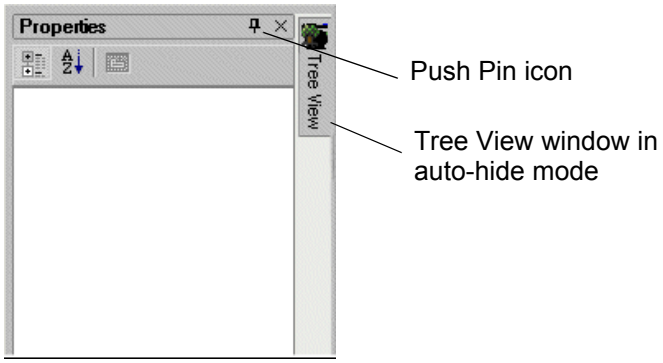
Auto hide minimizes a window and places a tab with the window name on the closest edge of the Micro Saint Sharp window. You can only place Tool windows in auto-hide mode.

To enable auto hide:

1. Right-click the title of the window, such as Tree View or Properties.
2. Select **Auto Hide** from the menu that displays.

   You can alternatively click the push pin icon on the title bar of the window. The icon points to the left when auto hide is enabled.

   When you move the mouse and the focus to another window, the window in auto-hide mode minimizes and a tab displays on the edge of the Micro Saint Sharp window with its name.



Displaying Windows in Auto-Hide Mode

Windows in auto-hide mode display as a tab along the edge of the Micro Saint Sharp main window with the name of the window.

To display a window in auto-hide mode:

Move the cursor over the window tab.

The window slides back into view and is ready for use. When you click the mouse off the window, the window loses focus and automatically slides back to the tab on the edge of the main window.

Disabling Auto-Hide Mode

Disabling auto-hide mode returns the window to its previous position.

To disable auto-hide mode:

1. Move the cursor over the tab for the window so that the window redisplays.

2. Right-click the window title and select **Auto Hide** so that the check mark adjacent to it is cleared.

   You can alternatively click the push pin icon on the title bar of the window. The icon points in a downward direction when auto hide is disabled.

Moving Windows

To move any window:

1. Click and drag the title bar of the window to the new location while pressing the left mouse button.

2. Release the mouse button when the window is in the new location.

Resizing Windows

To resize any window:

1. Move the cursor over the border of a window until the cursor changes into a double-headed arrow.

2. Press the left mouse button and move the border to the new location.

Changing the Active Window

To make a window active, click anywhere in the window.

If the window is not currently displayed, do the following:

From the **View** menu, select **Windows**, and then select the window you want to make active. You can alternatively select the window from the list of windows in the Section tab.

Closing Windows

To close a window:

Click the ✕ in the upper right corner of the window or tab. You can alternatively right-click the title of the window and select **Hide** from the menu that displays.

Customizing the Micro Saint Sharp Toolbar

You can customize the toolbar in Micro Saint Sharp by moving the position of sections of the toolbar, displaying or removing toolbar sections, adding new toolbar sections, controlling whether buttons display as image, text, or image and text, adding and removing default buttons, and adding additional buttons for commands to any of the toolbar sections.

You can also customize the display of toolbars by setting them to have the look and feel of Office XP, Windows XP, or Windows Classic. In addition, you can change the foreground color, background color, and font of the toolbars.

The default configuration of the toolbar is divided into two sections. The sections are split by a Separator, a vertical bar consisting of short horizontal lines. You can manipulate each section separately.

- The File System section includes buttons to create a model, open a model, save a model, and cut, copy and paste selected nodes.
- The Execution section includes buttons to use in model simulation, such as checking for errors, starting, stopping, stepping, and halting model execution, clearing the data in the Output window, and controlling execution speed in real time mode.

If you purchased the Animator module, an additional section displays on the right side of the toolbar with the Animator buttons.

Moving Toolbar Sections

You can move or reposition any of the toolbar sections.

To move a toolbar section:

1. Move the mouse over the dashed vertical line to the left of the toolbar section.

   The cursor changes to two double-sided arrows.

2. Press the left mouse button and move the toolbar section to the new location.

   You can dock a toolbar to any side of the window.

3. Release the mouse button.

Displaying and Removing Toolbars

You can display or remove any of the toolbar sections from the Micro Saint Sharp main toolbar.

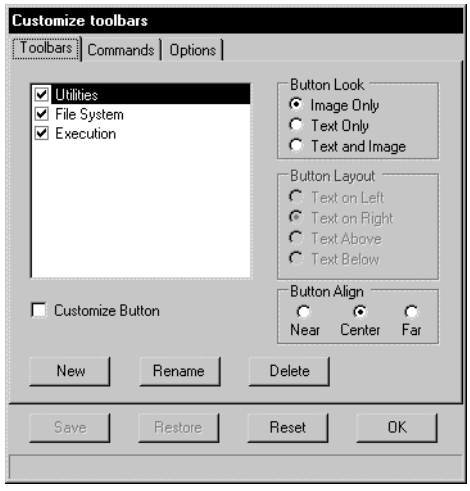
To display or remove a toolbar:

1. Click the down-facing arrow to the right of any toolbar section. The flyover text `Toolbar Options` displays.

2. Click **Add or Remove buttons**.

3. Click **Customize**.

   The Customize Toolbars dialog box displays.



4. Do any of the following:
   - To remove a toolbar, select the check box adjacent to the toolbar name to clear the check mark.

     Although you can delete a toolbar, removing the toolbar is recommended instead, in case you want to use the toolbar at a later time.
   - To add a toolbar, select the check box adjacent to the name of the toolbar so that a check mark appears.

     To include the Add or Remove Buttons and Customize command on the toolbar, select the **Customize Button** check box.
   - To reset the configuration to the original configuration, click **Reset**.

   The new toolbar configuration displays.

5. Click **OK** to close the Customize Toolbars dialog box.

Adding New Toolbars

You can add a new toolbar section to the Micro Saint Sharp toolbar and then add any buttons for commands that you might commonly use.

To add a new toolbar:

1. Click the down-facing arrow to the right of any toolbar section. The flyover text `Toolbar Options` displays.

2. Click **Add or Remove buttons**.

3. Click **Customize**.

   The Customize Toolbars dialog box displays.

4. Click the **New** button.

   The New Toolbar dialog box displays.

5. In the New toolbar name box, type a name for the toolbar and click **OK**.

   The new toolbar displays in the list of toolbars.

   You can now add buttons to the toolbar. For details, see "Adding Custom Toolbar Buttons" on page 54.

6. Click **OK** to close the Customize Toolbars dialog box.

Customizing the Display of Toolbar Buttons

You can customize the display of toolbar buttons by displaying an image only, an image and text, or text only.

To customize the display of toolbar buttons:

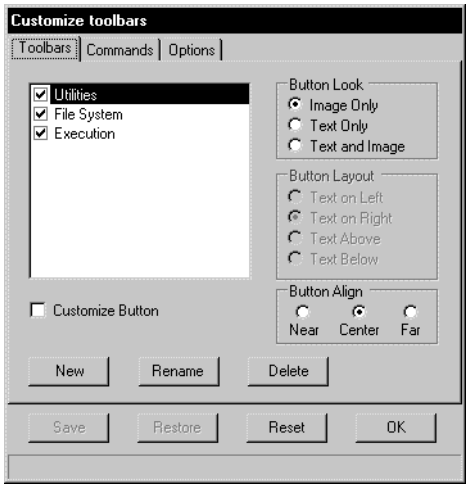
1. Click the down-facing arrow to the right of any toolbar section.

   The flyover text `Toolbar Options` displays.

2. Click **Add or Remove buttons**.

3. Click **Customize**.

   The Customize Toolbars dialog box displays.



4. Highlight the toolbar that has the buttons you want to change.

5. In the Button Look section, select one the following:
   - To display the button image only, select **Image Only**.
   - To display button text only, select **Text Only**.
   - To display the button image and text, select **Text and Image**.

     For the Text and Image selection, select the layout for the text and image in the Button Layout section: Text on Left, Text on Right, Text Above, or Text Below.

   The new toolbar configuration displays.

6. Click **OK** to close the Customize Toolbars dialog box.

Removing Default Toolbar Buttons

You can remove any of the default toolbar buttons.

To remove a default toolbar button:

1. Click the down-facing arrow to the right of any toolbar section.

   The flyover text `Toolbar Options` displays.

2. Click **Add or Remove buttons**.

   A list of buttons on the toolbar displays.

3. Select the button to remove.

   The button is removed from the toolbar. In the list of buttons, the check mark adjacent to the button name is also removed.

Adding Default Toolbar Buttons

You can add any of the default toolbar buttons after removing them.

To add a default toolbar button:

1. Click the down-facing arrow to the right of any toolbar section.

   The flyover text `Toolbar Options` displays.

2. Click **Add or Remove buttons**.

   A list of buttons on the toolbar displays.

3. Select the button to add.

   The button is added to the toolbar. In the list of buttons, a check mark displays adjacent to the button name. The button will be placed at the end of the toolbar. To place the button in its original position, click **Reset**.

Adding Custom Toolbar Buttons

You can add additional buttons to the toolbars for any of the Micro Saint Sharp commands.

To add custom toolbar buttons:

1. Click the down-facing arrow to the right of any toolbar section.

   The flyover text `Toolbar Options` displays.

2. Click **Add or Remove buttons**.

3. Click **Customize**.

   The Customize Toolbars dialog box displays.

4. Click the Commands tab.

   The list of Micro Saint Sharp menus displays on the left side of the tab in the Categories box. When you select a menu, the associated commands display on the right in the Commands box.

5. In the Categories box, select the menu containing the command to add to the toolbar.

6. In the Commands box, select the command and drag it out of the dialog box to the corresponding toolbar on the Micro Saint Sharp toolbar.

   The button displays on the toolbar. Flyover text describing the command is automatically associated with the button.

✓ Note:
To include the Add or Remove buttons and Customize command on a toolbar, click the Toolbars tab, select the toolbar name, and then select the Customize button.

---

7. Click **OK** to close the Customize Toolbars dialog box.

Removing Custom Toolbar Buttons

You can remove any of the custom toolbar buttons that you added.

To remove custom toolbar buttons:

1. Click the down-facing arrow to the right of any toolbar section.

   The flyover text `Toolbar Options` displays.

2. Click **Add or Remove buttons**.

3. Click **Customize**.

   The Customize Toolbars dialog box displays.

4. Click the Commands tab.

   The list of Micro Saint Sharp menus displays on the left side of the tab in the Categories box. When you select a menu, the associated commands display on the right in the Commands box.

5. On the toolbar, drag the button to the Commands box.

   The button is removed from the toolbar.

6. Click **OK** to close the Customize Toolbars dialog box.

Resetting the Toolbar Configuration

You can reset the toolbar configuration at any time to the original configuration.

To reset the toolbar configuration:

From the **View** menu, select **Reset Toolbars**.

The original toolbar configuration is restored.

Customizing the Look and Feel of Toolbars

You can customize the display of toolbars by setting them to have the look and feel of Office XP, Windows XP, or Windows Classic. You can also change the foreground color, background color, and font of the toolbars.

To customize the look and feel of a toolbar:

1. Click the down-facing arrow to the right of any toolbar section.

   The flyover text `Toolbar Options` displays.

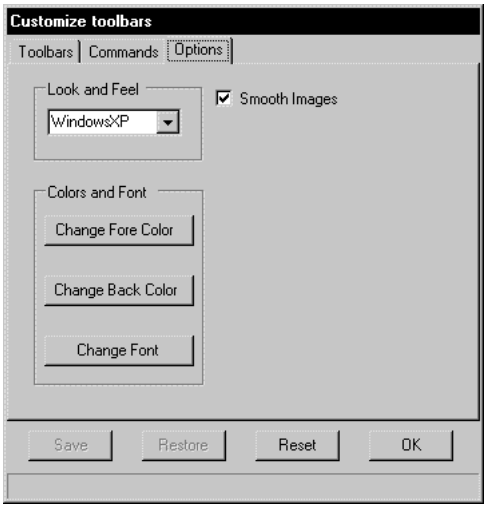
2. Click **Add or Remove buttons**.

3. Click **Customize**.

   The Customize Toolbars dialog box displays.

4. Click the Options tab.

   A list of toolbar options displays.



5. In the Look and Feel section, do any of the following:
   - To set the toolbar format to Office XP style, select Office XP. When you make this selection, you can also select the Smooth Images check box if you want smoothed images.
   - To set the toolbar look and feel to Windows XP style, select Windows XP.
   - To set the toolbar look and feel to Windows Classic style, select Classic.

6. In the Colors and Font section, do any of the following:
   - To change the foreground color, click the **Change Fore Color** button and select the color to use.
   - To change the background color, click the **Change Back Color** button and select the color to use for the background.
   - To change the font that is used, click the **Change Font** button and select the font to use for text.

7. To save the settings for future use, click **Save**.

   You can revert to the default configuration by clicking **Reset**.

8. Click **OK** to close the dialog box.

5

Basic Procedures

Basic Micro Saint Sharp procedures include creating, opening, and saving models, and exporting model results. Model results that you can export include task data information, queue data information, and snapshot data files. You can also search for and replace text anywhere in the model.

Working with Models

A Micro Saint Sharp model consists of the task network diagram, and the variables, functions, data collection snapshots, scenario events, entity attributes, charts, watches, and execution settings. All of these components are saved with each model and you can export many of them in `.xml` format so you can view or work with them in other applications.

Creating Models

After opening Micro Saint Sharp, you begin work by creating a new model or opening an existing one.

To create a new model:

1. If the task network diagram is not displayed, select **Window** from the **View** menu, and then select **Task Network**.

   A blank network diagram displays. A new model is assigned the name `Untitled.saint` until you save it with a new name.

2. From the **File** menu, select **Save Model As**, or click the **Save** button on the Main toolbar.

   The Save Model As dialog box displays.

3. If necessary, change the directory where you want to save the file.

4. In the File name text box, type a file name and click **Save**.

   Micro Saint Sharp automatically assigns model files the extension `.saint`.

✓ Note:
You can also use the Save As command to make a backup copy of a model.

Opening Existing Models

You can open any existing model in a few simple steps. Micro Saint Sharp lists the four models you have most recently opened under the Recent Files command.

To open a recent model:

1. From the **File** menu, select **Recent Models**.

   The last four models appear in chronological order, with the most recently opened file at the bottom of the list.

2. Select the model.

   The model opens.

To open a model not on the list:

1. From the **File** menu, select **Open Model** or click the **Open Model** button on the Main toolbar.

   The Open Task Network dialog box displays, listing the models in the current directory.

2. From the Files of type list, make sure that Saint files is selected.

   Only the files of the type you select display.

3. If necessary, change to the directory containing the model.

4. Select the model from the listing of models or type the name of the file in the File name box.

5. Click **Open**.

   Micro Saint Sharp opens the model you selected.

> ✓ Note:
> You can only open one model at a time in Micro Saint Sharp, but you can open multiple instances of Micro Saint Sharp.

Opening Models From Previous Versions of Micro Saint

You can open Micro Saint models in Micro Saint Sharp that were created in Micro Saint versions 4.0 and 4.1. Micro Saint Sharp will automatically convert the models to the new format.

To open a model from a previous Micro Saint version:

1. From the Micro Saint Sharp **File** menu, select **Open**.

   The Open dialog box displays.

2. From the Files of Type list box, select `.mod`.

3. Locate and select the file.

4. Click **Open**.

   The file is converted to Micro Saint Sharp format and displays in the Micro Saint Sharp window. The conversion may take a few minutes depending on the complexity of the model.

Saving Models

As you work on a model, save frequently—do not wait until you finish working. By saving the model frequently, you reduce the amount of work lost in the event of a power failure or other unforeseen problem.

To save a model:

From the **File** menu, select **Save Model** or click the **Save Model** button on the Main toolbar.

Micro Saint Sharp saves the model and overwrites any existing model with the same name.

Note:
Micro Saint Sharp automatically saves a backup copy of the current model every two minutes to the Micro Saint Sharp installation folder. The model is named `autosavebackup.saint`.

Exporting Model Information

You can export result data from a Micro Saint Sharp model, including the following:

- Results/QueueDataCollection includes the queue information. Queue information files record the queue ID, queue name, number of times an entity entered the queue, the minimum time, maximum time, and mean time/standard deviation that an entity spent in the queue, and the minimum time, maximum time, and mean time/standard deviation that an entity spent waiting in the queue before being allowed to execute the task.

- Results/TaskDataCollection includes the task information. Task information files contain the task number, task name, minimum task time, maximum task time, mean task time and standard deviation, and number of times a task was performed.

- Results/*snapshotname* includes the snapshot information. A different file is created for each snapshot.

All of the files except for the snapshot files are exported as `.xml` files. You can export snapshot files as `.xml` or `.res` (tab delimited text) files.

■ Exporting Model Information

To export Micro Saint Sharp results:

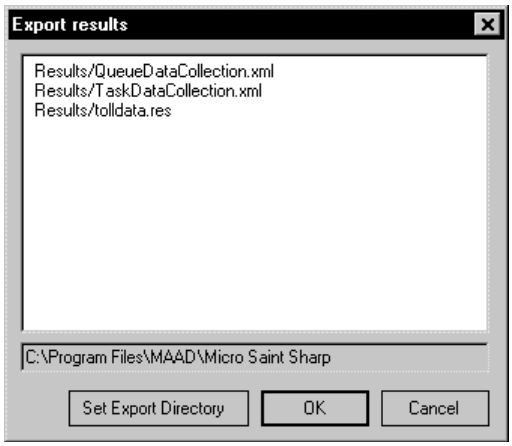
1. In the Execution Settings dialog box, select the Output type for the snapshot files (see "Editing Execution Settings" on page 159).

   To create `.res` files (ASCII text files), select **Tab Delimited**. To create `.xml` files, select **XML**.

   For details on execution settings, see "Execution Setting Parameters" on page 159.

2. From the **File** menu, select **Export Results**.

   The Export results dialog box displays.



3. Click the **Set Export Directory** button and select the location to store the results.

4. Select the files you want to export.

   Use Shift-click or Ctrl-click to select multiple files.

5. Click **OK**.

   The files are exported.

> ✓ Note:
> Snapshot files can also automatically be exported after the simulation. For details, see the Snapshot “Properties Tab” on page 147.

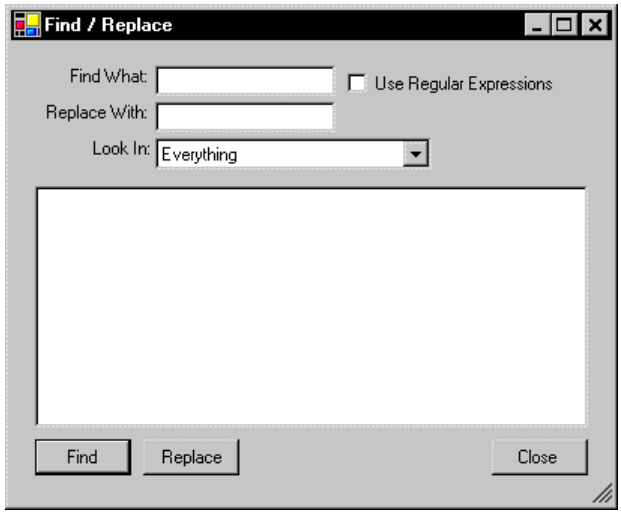
Searching for Text

The Find command locates a string of text in the model. You can search the entire model or selected components only: tasks, functions, scenario events, snapshots, variables, or entity variables.

To search for text:

1. From the **Edit** menu, select **Find**.

   The Find/Replace dialog box displays.



2. In the Find What box, type the text string to search for.

   The search feature is case-sensitive so consider capitalization when you enter the string (for example, Time is different than time). Space characters are allowed in the string.

3. In the Look In section, select the model elements to search.

   Options include Everything, Tasks, Functions, Scenario Events, Snapshots, Variables, and Entity Variables.

4. Click **Find**.

   Micro Saint Sharp searches for the text string and displays each occurrence in the lower part of the Find/Replace dialog box.

5. Click **Close** to close the Find/Replace dialog box.

Replacing Text

To change a string of text in the model to a different string of text, you use the Find command. You can search the entire model or selected components only: tasks, functions, scenario events, snapshots, variables, and entity variables.

To replace text:

1. From the **Edit** menu, select **Find**.

   The Find/Replace dialog box displays.

2. In the Find What box, type the text string to search for.

   The search feature is case-sensitive so consider capitalization when you enter the string (for example, Time is different than time). Space characters are allowed in the string.

3. In the Replace With box, type the text you want to use as a replacement.

4. In the Look In section, select the model elements to search.

   Options include Everything, Tasks, Functions, Scenario Events, Snapshots, Variables, and Entity Variables.

5. Click **Find**.

   Micro Saint Sharp searches for the text string and displays each occurrence in the lower part of the Find/Replace dialog box.

6. Click **Replace**.

   Micro Saint Sharp replaces the text and displays the new string of text.

7. Click **Close** to close the Find/Replace dialog box.

■ Replacing Text

6

Network Diagram

The graphical display of a Micro Saint Sharp model shows the tasks, networks, groups, comments, queues, decision nodes, and paths connecting these items. Once you have analyzed the modeling process, drawing the task network diagram in Micro Saint Sharp is relatively easy. Using the tools on the Task Network palette, you can place the tasks, networks, comments, and groups onto the network diagram and then draw the paths connecting them. You can then designate the starting task, and define the task, network, queue, and decision logic for each task.

This chapter describes how to create a network diagram. For details on defining the objects in the diagram, see Chapter 7: “Model Components.”

Creating a Network Diagram

To create a network diagram:

1. If the task network window does not currently display, select **Window** from the **View** menu and then select **Task Network**.

   A blank network diagram displays. A new model is assigned the name `Untitled.saint` until you save it with a new name.

2. Add the network components and draw the paths connecting the components.

   For details, see "Adding Network Objects" on page 71 in the following section and "Drawing Paths" on page 75.

3. Open the Description dialog box or Properties window for each item and enter the defining parameters.

   Parameters are described in Chapter 7: "Model Components."

Working With Network Objects

Network objects include the tasks, networks, comments, and groups available on the tool palette. For detailed descriptions of these objects, see Chapter 7: "Model Components."

Adding Network Objects

You can add objects in two ways.

To add a network object using the task network palette:

1. Click the Task Network palette.
2. Select the item you want to add.

   When you select an item, a green box displays around the item.
3. Click the left mouse button and drag the item to the task network diagram while pressing the button.
4. Release the left mouse button.

   The object displays on the network diagram.
5. Double-click the object.

   A dialog box displays in the network diagram for you to enter the defining properties. You can perform this step at any time. When you double-click a network, the network diagram displays and the associated Properties window displays for the network parameters.

To add a network object directly in the task network diagram:

1. Right-click the mouse on the task network diagram.
2. Select one of the following from the menu that displays: **Add Task**, **Add Group**, **Add Comment**, or **Add Network**.

   The object displays on the network diagram.

3. Double-click the object.

   A dialog box displays in the network diagram for you to enter the defining properties. You can perform this step at any time. When you double-click a network, the network diagram displays and the associated Properties window displays for the network parameters.

Deleting Network Objects

You can delete any of the network objects. If you delete a task or network, the associated path is automatically deleted.

To delete a network object:

1. Click the object to delete.

   When you select an object, a green box displays around the object.

2. Press the Del key on the keyboard. You can alternatively right-click the mouse and select **Cut** from the menu that displays.

Using Grids

You can enable an invisible grid that attracts network objects to it. As you move objects on the network diagram, they snap or "jump" to the invisible grid. When the snap to grid feature is disabled, you can move objects in a continuous line. Snap to grid aids you in aligning the objects on the network diagram.

To enable snap to grid:

From the **Utilities** menu, select **Enable Snap to Grid**.

The Snap to Grid feature is enabled and a check mark displays adjacent to the Snap to Grid option in the Utilities menu. As you move objects on the network diagram, they snap to the invisible grid.

To disable snap to grid:

From the **Utilities** menu, select **Enable Snap to Grid**.

The Snap to Grid feature is disabled and the check mark adjacent to Snap to Grid item is removed. As you move objects on the network diagram, they move in a continuous line.

Moving Network Objects

Once placed, you can move any of the network objects on the network diagram. Any associated paths automatically move and adjust to the new location.

To move a network object:

1. Click the object once with the left mouse button.

   A box displays around the object to show it is selected.

2. Click the object again and while pressing the left mouse button, move the object to the new location.

Aligning Network Objects

You can align network objects with one another along a vertical line or along a horizontal line. You can also center objects vertically or horizontally. Micro Saint Sharp aligns the objects with the leftmost, rightmost, highest, or lowest object.

To align network objects:

1. Select the objects to align.

   You can either click the mouse and drag the mouse to create a box around the items or press the Shift or Ctrl key and click the items with the mouse.

2. From the **Utilities** menu, select **Alignment**. You can alternatively right-click the mouse and select **Alignment**.

3. Select one of the following:
   - To align the left sides, select **Align Left**. Objects are aligned with the leftmost object.
   - To align the right sides, select **Align Right**. Objects are aligned with the rightmost object.
   - To align the tops, select **Align Top**. Objects are aligned with the highest object.
   - To align the bottoms, select **Align Bottom**. Objects are aligned with the lowest object.
   - To align the objects on the same horizontal line, select **Center Horizontally**. Objects are aligned on a horizontal line that is midpoint between the highest and lowest object.
   - To align the objects on the same vertical line, select **Center Vertically**. Objects are aligned on a vertical line that is midpoint between the leftmost and rightmost object.

Using Auto Layout

The Auto Layout feature arranges your network diagram to it's most orderly or symmetrical fashion. This includes aligning objects, providing a layout where there is minimum cross over of path lines, and similar features. The GoLayout algorithm is used (www.nwoods.com).

To use auto-layout:

From the **Utilities** menu, select **Auto Arrange Network**.

Micro Saint Sharp rearranges the network diagram.

Drawing Paths

Paths connect tasks and networks on the network diagram. Paths control the flow of execution during the simulation.

To draw a path:

1. Do one of the following:
   - For a task, move the cursor over the decision icon for the task that is the start point for the path. The decision icon is shaped like a diamond and displays on the right side of the task icon.
   - For a network, move the cursor over the right-most edge of the network icon that is the start point for the path.

   The cursor changes from a pointer to a hand.

2. Left-click the mouse, and while pressing the mouse button, draw the path to the destination task or network.

3. Release the mouse button.

   The path displays. The path is marked with an arrow at the endpoint to show the path direction.

✓ Note:
To draw a path that re-enters the task, click the decision icon for the task. A circular path displays at the top of the task icon.

Displaying Path Logic

You can display the dialog box that describes the logic or code for a path. Path logic is contained in the Tasks Paths tab. For additional details, see “Paths Tab” on page 106.

To display path logic:

1. Click the path.

   Small green squares and diamonds display on the path. Squares outline each line segment and diamonds display at the beginning and end of the path.

2. From the **File** menu, select **Edit**. You can alternatively right-click the mouse and select **Edit** from the menu that displays.

   The Paths Description dialog box for the path opens and displays the path logic.

Removing Paths

To remove a path:

1. Click the path.

   Small green squares and diamonds display on the path. Squares outline each line segment and diamonds display at the beginning and end of the path.

2. Press the Del key on the keyboard. You can alternatively right-click the mouse and select **Delete** from the menu that displays.

   The path is removed.

Designating the Start Node

You must select one of the tasks or networks as the start node. The start node controls where the simulation begins. You can also designate the start node in the Execution Settings dialog box. For details, see "Execution Setting Parameters" on page 159.

To designate the start node:

1. On the network diagram, select the task or network to use as the starting point.

   When you select an object, a green box displays around the object.

2. From the **Execution** menu, select **Set Start Object**. You can alternatively right-click the mouse and select **Set Start Object** from the menu that displays.

   The outline of the object is bolded to show that it is the start node.

Zooming the Network Diagram

Using the zoom feature of Micro Saint Sharp, you can view a specific part of the diagram in detail or broaden the diagram view.

To zoom the network diagram:

From the **View** menu, select **Zoom** and then select the magnification level.

Magnification levels vary from 10 to 800%.

✓ Note:
You can also zoom in and out of the network diagram by pressing CTRL and scrolling the mouse wheel one click at a time.

Panning the Network Diagram

Panning displays areas of the network diagram that are not currently displayed. When panning, you are not restricted to moving the mouse in a horizontal and vertical direction, but you can move the mouse diagonally.

To pan the network diagram:

1. Press the space bar.

   The cursor takes the following shape .

2. Keep pressing the space bar and move the mouse in the desired direction.

3. When you reach the position you want, release the space bar.

   You can also pan the display by moving the cursor in the relative position box in the Task Network palette or by using the scroll bars.

Jumping to Network Objects

Jumping to a network object displays and selects the object in the network diagram.

To jump to a network object:

1. In the task network tree view, right-click the object.

2. Select **Jump to Object in Network** from the menu that displays.

   The network diagram redraws so that the object displays. The object is automatically selected.

Changing Network Levels

To move down a network level:

On the task network diagram, double-click the network.

A new task network window opens displaying the network.

To move up a network level:

Click the Task Network or Network tab at the top of the network diagram window.

The selected network level displays.

Opening Another View of the Network Diagram

You can open another window with the network diagram so that you can have two different view of the same task network open at the same time. This can be particularly useful when you are working with large or complicated diagrams, want to open some grouped objects, or you would like to have different parts of the diagram at different zoom levels.

To open another task network view:

In the network diagram, right-click the mouse and select **Add Task Network View** from the menu that displays.

A second task network diagram window opens.

Editing Text and Objects

With Micro Saint Sharp tools you can select text or a network diagram object, such as a task, path, network, comment, or group. Once selected, you can copy, delete, and paste text and objects.

Selecting Text or Objects

Before you can edit text or objects, you must first select them. You can select a text string or an object such as a task or network. How you select an item to edit depends on whether the item is a text string or an object on the network diagram.

To select a text string:

First open the Description dialog box for the network diagram object containing the text string. Then do one of the following:

- To select a string of text, click with the mouse and drag the mouse through the text.
- To select a word, double-click the word.

To select a network object:

- Click the item once with the left mouse button.

  A box displays around the object to show it is selected. Paths leading out of the object are highlighted in red and paths leading into the object are highlighted in blue. When you select a path, small triangles display at the beginning and endpoints of the paths; small squares display outlining each line segment.

- To select multiple items on the network diagram, click with the mouse and drag the mouse to create a box around the items. You can alternatively press the Shift or Ctrl key and click the items with the mouse.

Copying Text and Objects

The Copy command copies the selected text or object to the Windows clipboard. Copying an object copies all of the object's properties. Once the text or object is on the clipboard, you can use the Paste command to insert it elsewhere in the model or into another application (see "Pasting Text and Objects" on page 82).

To copy text or an object:

1. Select the text or objects to copy.

2. From the **Edit** menu, select **Copy**. You can alternatively right-click the mouse and select **Copy**.

Copying the Graphical Network Diagram

You can copy an entire network diagram to the Windows clipboard and paste the copy into another application. Only the graphical depiction of the network is copied. You cannot paste the diagram into Micro Saint Sharp.

To copy a network diagram:

1. Open the model to copy.

2. From the **Edit** menu, select **Copy Network Diagram**.

   The model is copied to the clipboard. You can now paste the contents into another application.

Cutting Text and Objects

You can cut or remove text and objects and place them on the Windows clipboard. Cutting removes the text or object and places it on the clipboard. Once the item is on the clipboard, you can use the Paste command to place it elsewhere in the model or in another application.

To cut text or an object:

1. Select the text or object to cut.

2. From the **Edit** menu, select **Cut**. You can alternatively right-click the mouse and select **Cut**.

   The text or object is removed.

Deleting Text and Objects

You can delete selected text and objects without placing them on the clipboard. Use the Del key on the keyboard to delete something that you do not plan to reuse, especially if the clipboard currently contains an item that you do not want to lose.

To delete text or an object:

1. Select the text or object to delete.

2. Press the Del key on the keyboard. You can alternatively right-click the mouse and select **Cut**.

   The text or object is deleted.

Note:
If you accidentally delete text, press Ctrl-Z to retrieve it.

Pasting Text and Objects

You can paste whatever is copied to the clipboard with the Paste command. If the clipboard is empty, or if the current clipboard item is not in a format appropriate for the current location, the Paste command appears dimmed so that you cannot select it. You cannot use the Paste command to replace a network that is currently displayed or any network above it.

To paste text or an object:

1. From the **Edit** menu, select **Paste** or click the **Paste** button on the Main toolbar.

   The pasted text or object displays. For objects on the network diagram, the pasted object is on top of the copied object. Micro Saint Sharp automatically assigns a new ID to the item, but the name is the same as the copied item.

2. For items on the network diagram, select the pasted item and use the mouse to move it to the new location.

Displaying Network Object Properties in HTML

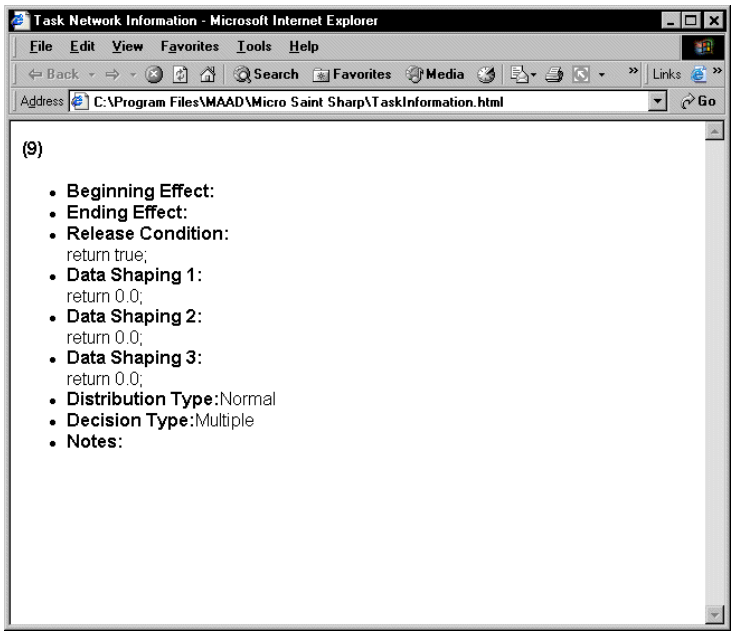
You can select any network object or item in the tree view and display the properties in HTML format. For tasks, the beginning effect, ending effect, release condition, distribution information, decision information, queue information, and notes display. For networks, the tasks and any notes display.

To display network object properties in HTML:

1. Do one of the following:
   - For an object on the task network diagram, select the object.
   - For variables, functions, snapshots, scenario events, entity attributes, charts, and watches select the item in the tree view.

2. Right-click the mouse and select **Show Properties as HTML** from the menu that displays.

   Your default web browser automatically opens and displays the properties.



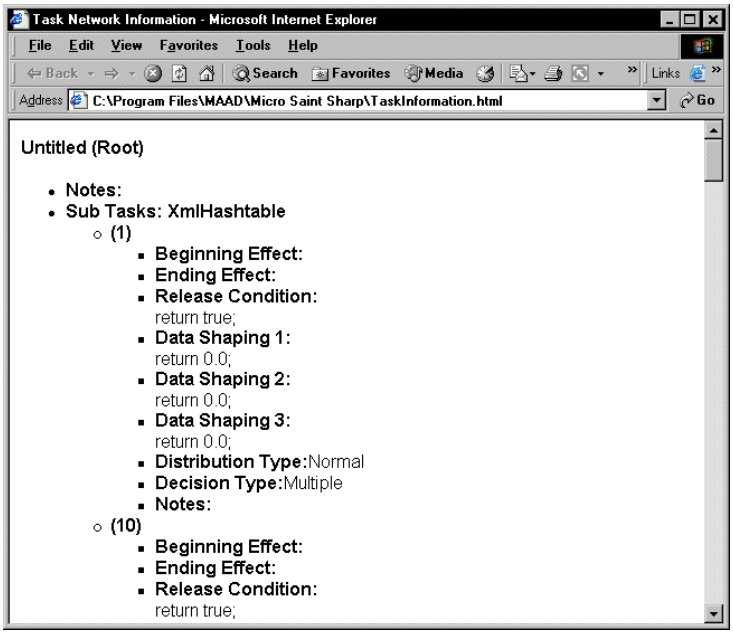
Displaying Task Network Properties in HTML

You can display the properties of the task network model in HTML format. Network notes, task properties, variables, functions, snapshots, scenario events, entity attributes, watches, and charts display.

To display task network properties in HTML:

From the **Utilities** menu, select **View Model as HTML**.

Your default web browser automatically opens and displays the model properties.



Printing the Task Network Diagram

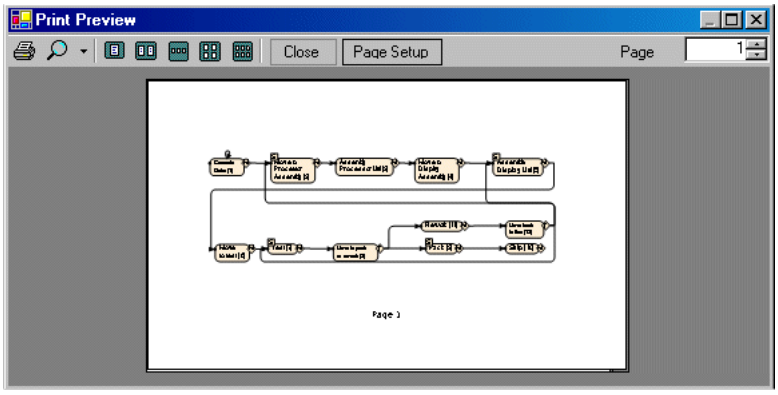
Before printing the task network diagram, Micro Saint Sharp displays a preview of how the task network diagram will appear when printed. From this window, you control the zoom factor, the number of pages that are viewed, and can add header information.

For additional task network printing parameters, see “Misc Tab” on page 165.

To print the task network diagram:

1. From the **File** menu, select **Print Network**.

   The Print Preview window displays.



2. Do any of the following:
   - To zoom the view, click the down arrow adjacent to the Magnifying glass icon and select the zoom level.
   - To view more than one page at a time when the task network diagram extends over more than one page, click one of the buttons to the right of the Magnifying glass: one, two, three, four, or six pages.
   - To add a header to the task network diagram, click the Page Setup button and type the header information. The header information prints on each page.

- To change the size that the diagram is printed, click the **Page Setup** button and select the size from the Zoom list box.
- To scroll through the pages when the task network extends over more than one page, click the Up and Down arrows adjacent to the right of the Page box in the upper right corner of the Print Preview window.
- To close the Print Preview window, click the **Close** button.

3. When you are ready to print the task network diagram, click the **Print** button on the Print Preview window.

■ Printing the Task Network Diagram

7

Model Components

The graphical display of a Micro Saint Sharp model shows the networks, groups, tasks, queues, decision nodes, and paths connecting these items. You can also use comments to add headers for clarification, add other text to the task network diagram, and display values of variables on the task network diagram as the model runs. In addition to the graphical task network diagram, Micro Saint Sharp uses variables, functions, snapshots, and scenario events in the model simulation. All of these items are defined in description dialog boxes or in the associated Properties window.

Networks

When a task in the network is very complex, you may want to simplify it by dividing it into more specific tasks. You can do this by defining the task as a subnetwork. A subnetwork can contain all the same elements as the main network—tasks, paths, decisions, queues, and subnetworks at a still lower level—and is defined by its own network diagram. In addition, a subnetwork can have a release condition similar to a task, and it can be preceded by a queue. Subnetworks are represented by a rectangular shape on the task network diagram.

A list of networks associated with the model displays in the tree view below the Task Network node. You can display network descriptions, and add, edit, and delete networks.

Note:
In some cases, you may want to group objects instead of creating a new network. For details, see “Groups” on page 95.

Displaying Network Descriptions

To display a network description:

On the task network diagram, double-click the network. You can alternatively right-click the network in the tree view and select **Object Description** from the menu that displays.

The Properties window for the network displays with a list of the network parameters.

Adding Networks

A network can contain other networks within it. In a model, all networks are subnetworks except for the top network.

Note:
All incoming tasks in a subnetwork display in blue and all outgoing tasks display in red.

To add a network:

1. From the Task Network palette, drag the Network tool to the correct place on the task network diagram. You can alternatively right-click the mouse on the network diagram and select **Add Network** from the menu that displays.

   Micro Saint Sharp automatically numbers the network and adds it to the tree view below the Network node.

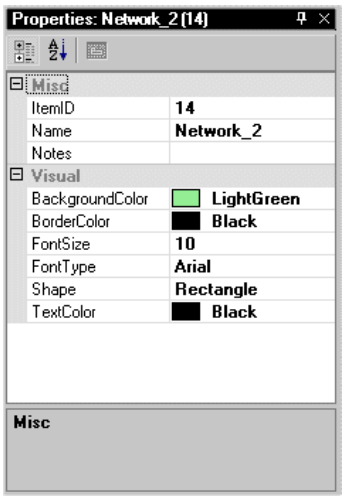
2. To display the network in the task network diagram, double-click the network icon in the task network diagram.

3. To edit the network properties, use the Properties window for the network.

   Network parameters are described in the following section.

Network Parameters

Network parameters consist of the ItemID, name, notes, and parameters controlling the appearance of the network icon.



- **ItemID**

  Contains the number of the network. This number is automatically assigned by Micro Saint Sharp but you can change it by entering a new value. ID numbers must be unique.

- **Name**

  Contains the name of the network. To change the name, click in the Name box and type the name of the network.

- **Notes**

  Contains any notes about the network. Entering notes is optional, but notes can be useful for future reference. To add notes, click in the Notes box, and then click the down-facing arrow that displays. Then type the notes in the text box.

- **BackgroundColor**

  Controls the background color of the network icon. To change the color, click in the BackgroundColor box, and then click the down-facing arrow that displays. Then select the new color.

- **BorderColor**

  Controls the border color of the network icon. To change the border color, click in the BorderColor box, and then click the down-facing arrow that displays. Then select the new color.

- **FontSize**

  Displays the font size for the network name. To change the font size, click in the FontSize box and type the new size. The network icon changes in size to accommodate the new font size.

- **FontType**

  Controls the type of font used for the network name. To change the font, click in the FontType box, and then click the down-facing arrow that displays. Then select the new font.

- **Shape**

  Controls the shape of the network icon. To change the shape of the network icon, click in the Shape box, and then click the down-facing arrow that displays. Then select the new shape.

- **TextColor**

  Controls the color of the text used for the network name. To change the color, click in the TextColor box, and then click the down-facing arrow that displays. Then select the new color.

Editing Networks

To edit a network:

1. On the task network diagram, double-click the network.

   The network properties display in the Properties window.

2. Make the changes in the Properties window.

   For details on network parameters, see “Network Parameters” on page 92.

Deleting Networks

To delete a network:

1. On the task network diagram, select the network.

   When you select an item, a green box displays around the item.

2. Press the Del key on the keyboard.

   The network is removed. All associated tasks are also deleted.

Groups

Grouping objects allows you to spatially isolate tasks and networks together in a network without actually creating a network. For example, you could group a cell of machines together in a plant or group a series of tests without needing to create a network.

You can group objects in two ways: by selecting existing objects and grouping them or creating the group first and then adding the objects.

Grouping Objects by Selecting Existing Objects

To group objects by selecting existing objects:

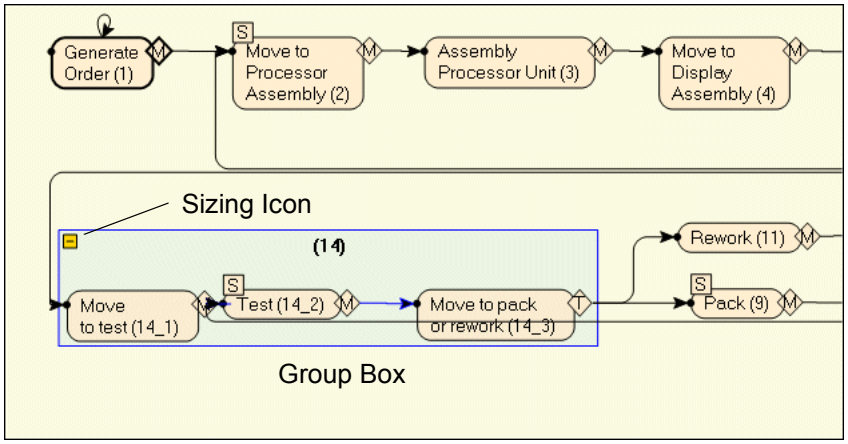
1. On the task network diagram, Shift-click the tasks to group.

2. From the **Utilities** menu, select **Group Objects**.

   A shaded box displays around the objects. A small icon with a minus sign displays in the upper left corner of the group box. To collapse the box to the size of a task icon, click the minus icon; the icon changes to a plus. To expand the box to the original size, click the plus icon.

   You can move the Group box on the task network diagram as if it were any other network element.

   The following illustration shows an example of a group.



Grouping Objects by First Creating a Group

To group objects by first creating a group:

1. Click the Task Network palette.
2. Click the Group tool.

   When you select an object, a green box displays around the object.
3. Click the left mouse button, and while pressing the button, drag the Group tool to the correct place on the task network diagram.
4. Release the left mouse button.

   The group icon displays on the task network diagram.

   You can alternatively add a group by right-clicking the mouse on the task network diagram and selecting **Add Group**.
5. To add tasks or networks to the group, drag the Task or Network tool from the Task Network palette into the group icon.

6. Using the mouse, move the objects within group boundaries to the correct location and draw the appropriate connecting paths.

   As you move objects, the boundaries of the Group automatically resize to contain the objects.

   A small icon with a minus sign displays in the upper left corner of the group box. To collapse the box to the size of a task, click the minus sign; the icon changes to a plus. To expand the box to the original size, click the plus icon.

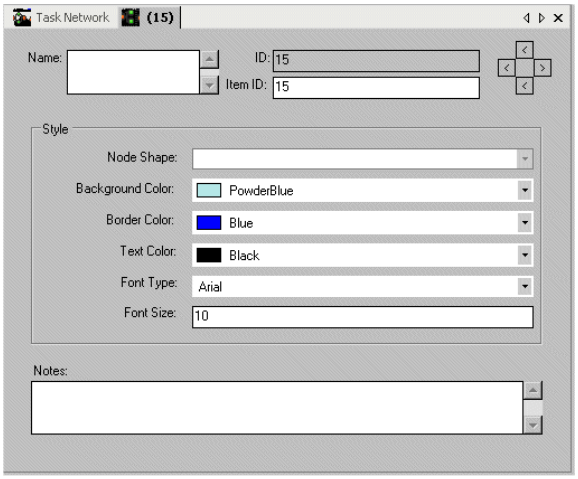
   You can move the Group box on the task network diagram as if it were any other network diagram element.

Setting Group Properties

To set group properties:

1. Double-click anywhere in the Group box.

   The Group Description dialog box displays.



2. Enter the Group parameters.

   Parameters are described in the following section.

Group Parameters

Group parameters consist of the name and ID for the group, the physical appearance of the group, and any notes.

Following are detailed descriptions of the group parameters.

- **Name**

  Contains the name for the group.

- **ID**

  Contains the Micro Saint Sharp-defined ID number for the group. Micro Saint Sharp automatically assigns the next available number when a new group is created. To change the ID, enter a new ID in the Item ID box.

- **Item ID**

  Contains the user-defined ID for the group. You can enter a different number.

- **Style section**

  Contains parameters controlling the physical appearance of the group.

  For details on these parameters, see “Network Parameters” on page 92.

- **Notes**

  Contains any notes about the group. Entering notes is optional, but notes can be useful for future reference.

Ungrouping Objects

You can ungroup the objects contained within a group.

To ungroup objects,

1. In the task network diagram, click the group.

2. From the **Utilities** menu, select **UnGroup Object**. Alternatively, you can right-click the mouse and select **Ungroup Objects** from the menu that displays.

   The tasks are ungrouped. You will need to reposition the tasks to the correct locations.

Tasks

A task is the basic building block of a model and contains timing information, execution constraints, effects of the task on the system, and routing information. A list of tasks associated with the model displays in the tree view below the Task Network node. You can display task descriptions, and add, edit, and delete tasks.

Displaying Task Descriptions

To display a task description:

On the task network diagram, double-click the task. You can alternatively right-click the task in the tree view and select **Object Description** from the menu that displays.

The Task Description dialog box displays in the main Document window and the associated Properties window for the task displays.

Adding Tasks

To add a task:

1. From the Task Network palette, drag the Task tool to the correct place on the task network diagram. You can alternatively right-click the mouse on the task network diagram and select **Add Task**.

   Micro Saint Sharp automatically numbers the task and adds it to the tree view below the Network node.

2. To display the task description, double-click the task icon in the task network diagram.

   The Task Description dialog box displays.

3. Enter the task parameters.

   Parameters are defined in the following section.

Task Parameters

The task name, ID, and whether data collection is enabled displays at the top of the Task Description dialog box. Additional parameters are contained on the Main, Timing, Paths, Queue, and Appearance and Notes tab.

Following are detailed descriptions of the parameters at the top of the Task Description dialog box.

- **Name**

  Contains the name of the task. Use a phrase that is descriptive of the task, for example, answer call or register patient.

- **ID**

  Contains the Micro Saint Sharp-defined ID number for the task. Micro Saint Sharp automatically assigns the next available number when a new task is created. The ID reflects the network hierarchy. For example, the first task in network 3 would have a default ID of 3_1. To change the last digit of the ID, enter a new ID in the Item ID box. ID numbers must be unique.

- **Item ID**

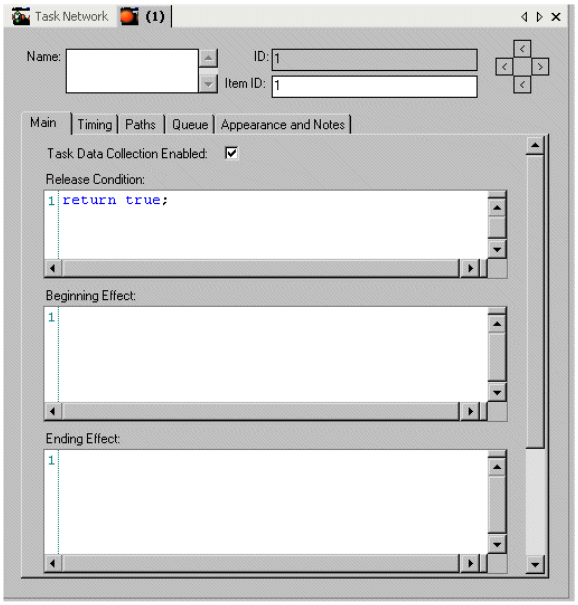
  Contains the user-defined ID for the task. You can enter a different number, but you can only use an ID that is not already in use.

- **Task Data Collection Enabled**

  Controls whether task data is collected. Select the check box to collect data on how long the task requires to execute. For data to be collected, you must also select **Enable Task Data Collection** in the Execution Settings dialog box (see “Enable Task Information Collection” on page 7-6).

Main Tab

The Main tab contains the release condition, beginning effect, ending effect, and launch effect for the task.



Following are detailed descriptions of the parameters in the Main task tab.

Note:
After you type the word Entity and the following period in an expression text box, Micro Saint Sharp automatically displays a list of system and user-defined entity attributes for selection. Double-click an attribute to select it.

♦ **Release Condition**

Contains an expression that determines whether the task can execute. A task can only execute when the release condition value is nonzero or true. The default value is `return true;`

You should exercise caution when adding expressions to a release condition for two reasons. First, a lockout condition can occur when a release condition is never met; the task is "Locked out" of the model. When this occurs, you receive a "Job can never be released" message when the model ends and a task is still waiting for its release condition to be met.

Second, it is very important that the release condition be used only for evaluating whether a job can be executed, rather than to initialize or set variables. This is because the release condition can be evaluated many times before a job is executed, causing all expressions in the release condition to be executed multiple times before the task actually is released, or started. For this reason, you should not use the release condition to increment variables or to initialize the `Entity.Tag` attribute. These sorts of expressions should be placed in the beginning effect.

- **Beginning Effect**

  Contains expressions indicating what happens when the task begins execution. The task execution time, or the `Entity.Duration`, is calculated after the beginning effect occurs. You must include a return statement in the statements defining the beginning effect. For details on return statements, see "Return Statements" on page 236.

- **Ending Effect**

  Contains expressions indicating what happens when the task finishes execution.

- **Launch Effect**

  The launch effect is used to execute expressions that cause Animator events (Animator is an optional add-on component). The launch effect is similar to a beginning effect except that it is evaluated after the `Entity.Duration` is calculated. The launch effect occurs only once.

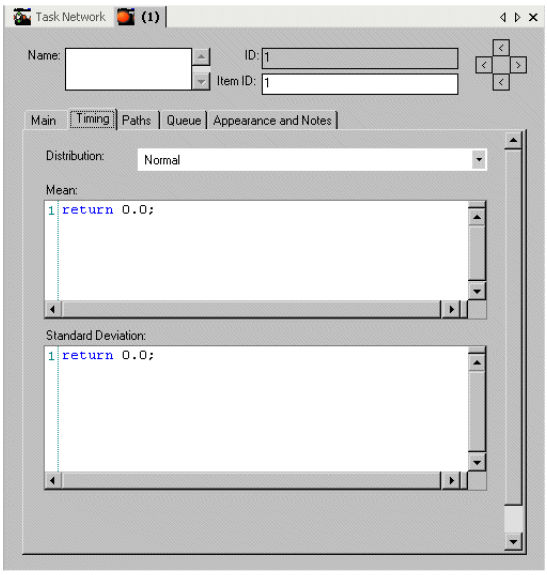
Micro Saint Sharp evaluates task expressions in the following order:

- Release condition
- Beginning effect
- Launch effect
- Ending effect

When you define the task expressions for a model, keep in mind that each expression can affect the expressions that come after it in the evaluation order. For example, if the mean time expression changes the value of a variable and that variable is used in the launch effect, the launch effect uses the changed value.

Timing Tab

The Task Timing tab contains the probability distribution used for the execution times. When you select a time distribution for a task, Micro Saint Sharp uses the distribution to generate random execution times that occur in the pattern predicted by the distribution. The Task Timing tab additionally contains the parameters used to calculate the distribution. These vary depending on the type of distribution.



Following are detailed descriptions of the parameters in the Timing tab.

- **Distribution**

  Contains a drop-down list of the statistical distributions provided by Micro Saint Sharp. For details about probability distributions, see Chapter 13: “Probability Distributions.”

- **Mean**

  Contains an expression or number for the mean time to be used in generating execution times. Time units can be seconds, minutes, hours, and so on. This field is used in conjunction with the Distribution. You must include a return statement in the expressions defining the mean time. For details on return statements, see “Return Statements” on page 236.

- **Standard Deviation**

  Contains the number or expression for the standard deviation (in time units) to use in calculations for the task.You must include a return statement in the expressions defining the standard deviation. For details on return statements, see “Return Statements” on page 236.

Different text boxes may display for other distribution-related parameters, depending on the type of distribution you select. For details on these parameters, see “Distribution Functions” on page 272.

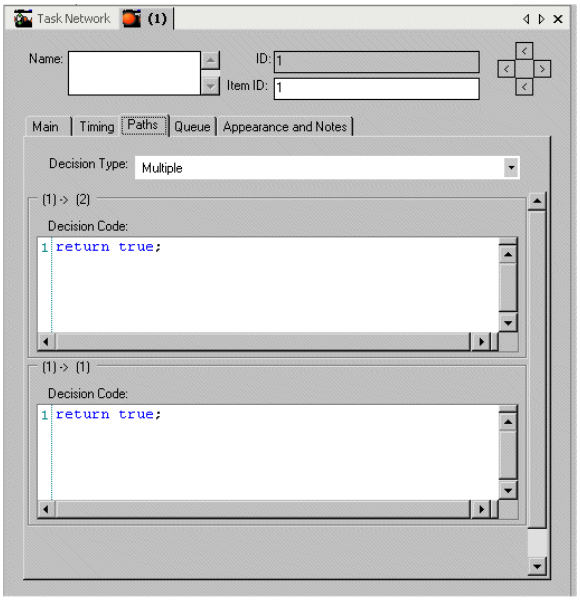
Note:
In the Properties window, the timing parameters are labeled as Data Shaping 1, 2, and 3.

Paths Tab

The Paths tab contains the Decision Type and Decision Code describing the conditions under which particular paths are taken.

Micro Saint Sharp uses three different decision types: tactical, probabilistic, or multiple. The decision type determines the path or paths that an entity should take. Decision nodes display with a diamond-shape in the task network diagram, and contain a letter inside of it (T, P, or M) indicating the type of decision.

Decision nodes are automatically added to a task by Micro Saint Sharp. Whenever a task has more than one possible path emerging from it, the logic for the Decision code determines the paths that the entity takes.



Following are detailed descriptions of the parameters in the Paths tab.

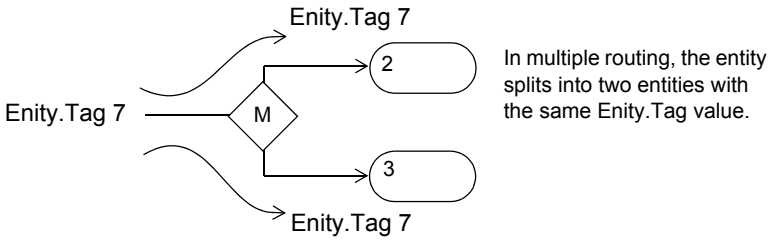
Note:
After you type the word Entity and the following period in any expression text box, Micro Saint Sharp automatically displays a list of system and user-defined entity attributes for selection. Double-click an attribute to select it.

♦ **Decision Type**

Contains the type of decision. Following are the available types:

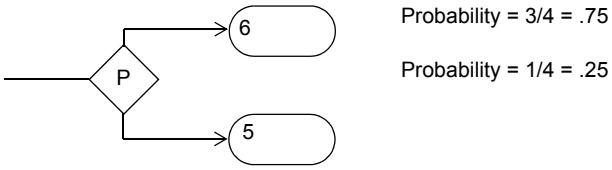
Multiple
All of the following jobs with nonzero routing conditions begin execution simultaneously following execution of the current task. When this happens, the entity exiting the current task splits into multiple entities, one for each following job.

The default decision type for a single path exiting a task is a Multiple decision type. This just means that all paths are exiting the task.

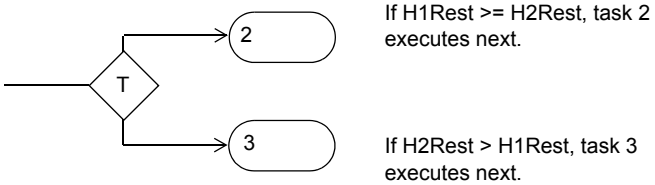


Probabilistic
Only one of the following jobs executes next; you supply the probability of each job executing next in its routing condition. The probability that a particular job follows is equal to its probability value in the Decision Code field divided by the sum of the probability values of all possible following jobs. For example, if the probability value is 25 percent for job A, you could enter .25, 1, or 25 in the Decision Code for job A and enter .75, 3, or 75, respectively, for the Decision Code for job B.

You can also enter an expression to define the probability, such as return `a*b + 1.36;`



Tactical
The job with the tactical expression that evaluates to the highest value executes next; you supply the tactical expression as the routing condition.



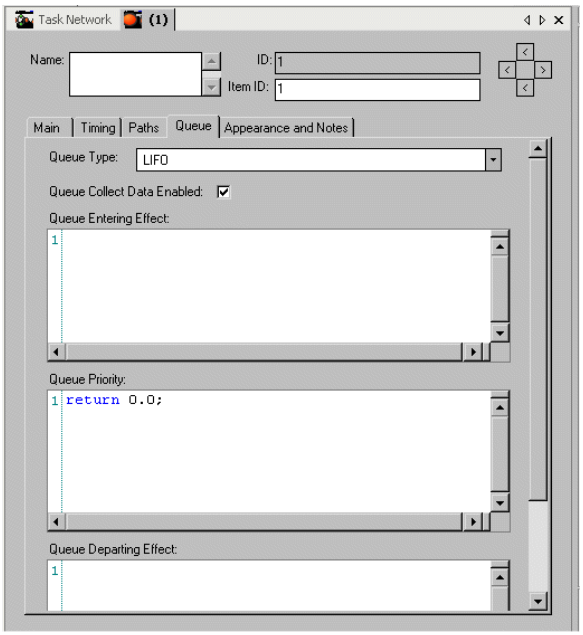
- **Decision Code**

  Contains the conditions under which particular paths are taken. You can express the routing condition as a number or an expression. The number and name of the next task to be executed when the routing conditions are met display above the Decision Code box. You must include a return statement in the expressions defining the decision code. For details on return statements, see “Return Statements” on page 236.

Queue Tab

The Queue tab contains the Queue Type, Queue Entering Effect, Queue Priority, and Queue Departing Effect for the queue associated with the task.

The queue is a waiting area in front of a task or network where entities accumulate while they are waiting to execute the job. Each time the release condition for the job is true, an entity can be released from the queue into the job. Entities are released in the order you specify when you define the queue. Queues are represented in the task network diagram as a square in front of the task.



Following are detailed descriptions of the parameters in the Queue tab.

Note:
After you type the word Entity and the following period in any expression text box, Micro Saint Sharp automatically displays a list of system and user-defined entity attributes for selection. Double-click an attribute to select it.

- **Queue Type**

  Specifies the order for sorting the entities. The following options are available:

  - None – entities wait until they are released. Entities are not sorted. This is the default value.
  - FIFO (First in First Out) – the next entity to be selected from the queue is the one that is in the queue the longest.
  - LIFO (Last In First Out) – the next entity to be selected from the queue is the entity that entered the queue most recently. This option represents a stacking order, so that the last one placed on top is the next one to be taken out.
  - Sorted – the next entity to be selected from the queue is based on a priority value that you assign. When you select this option, you must define the priority expressions identifying which entities have priority over other entities.

- **Queue Collect Data Enabled**

  Controls whether queue data is collected. Select this check box to enable queue data collection. You can collect data for as many queues as you like, but the more queues you enable, the larger the queue data file is when you run the model. To collect queue data, you must also select **Enable Queue Information Collection** in the Execution Settings dialog box (see “Enable Queue Information Collection” on page 7-7).

- **Queue Entering Effect**

  Contains the expressions indicating what happens when an entity enters a queue. Micro Saint Sharp evaluates the entering effect first followed by the queue priority. When you are defining the queue expressions for a model, keep in mind that each expression can affect the expressions that follow it in the evaluation order. The entering effect occurs only once.

- **Queue Priority**

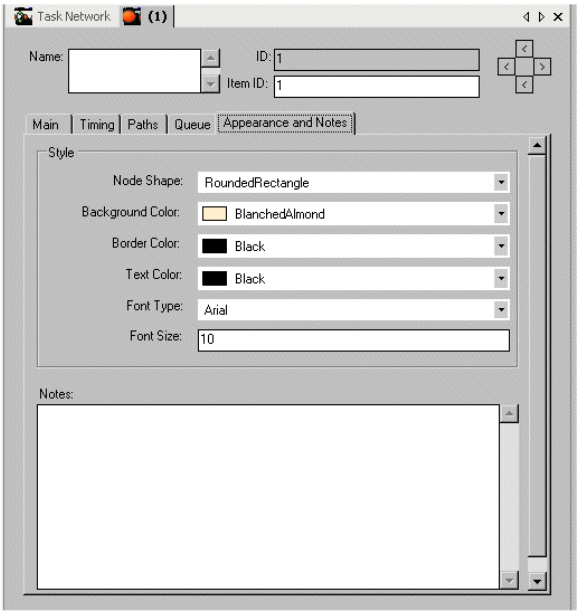
  Contains the expression used in sorting the entities in a queue. When it is time to release an entity from the queue, Micro Saint Sharp calculates the value of the priority expression for each entity in the queue. The entity with the highest priority value is released into the next job. For the priority expression to be meaningful, it must have different values for different entities, usually by involving the `Entity.Tag` attribute (see page 128). You must include a return statement in the expressions defining the priority. For details on return statements, see “Return Statements” on page 236.

- **Queue Departing Effect**

  Contains the expression indicating what happens when an entity leaves a queue. The departing effect occurs only once.

Appearance and Notes Tab

The Appearance and Notes tab contains the factors controlling the appearance of the task icon and a section for notes regarding the task.



Following are detailed descriptions of the parameters in the Appearance and Notes tab.

- **Style**

  Contains the parameters controlling the physical appearance of the task icon on the network diagram. For details on these items, see “Network Parameters” on page 92.

- **Notes**

  Contains any notes about the task. Entering notes is optional, but notes can be useful for future reference.

Using Multiple Entities

Generating Entities with Spinner Tasks—A common way to generate multiple tags to travel through a network is to use a spinner task. This task has two following tasks: the next task (task 2) and itself. The decision type is Multiple, so both paths are followed.

When task one finishes executing, the current entity splits into two entities, one of which goes on to the next task (or to a queue preceding the task) and one of which goes back to the beginning of task one. By adding one to the value of `Entity.Tag` as an ending effect of task one (in the statement `Entity.Tag++;`), you ensure that each entity that goes on to task two has a different `Entity.Tag` value. Task one keeps generating tags as long as the "if" expression you supply for routing back to task one is true.

Split Entities—Whenever you have a multiple decision that routes an entity to more than one following job, Micro Saint Sharp splits the entity into multiple entities that have the same `Entity.Tag` value as the original entity. If you are using `Entity.Tag` values as unique identifiers, you need to be sure either to renumber the "extra" entities, remove them from the system, or merge the split entities later on.

To merge the split entities, route them back to the same task. Specify a release condition for the task that requires both entities to be waiting for the task before it can be released (this condition often involves an array variable). Micro Saint Sharp then merges the two entities back into one.

Editing Tasks

To edit a task:

1. On the task network diagram, double-click the task.

   The Task Description dialog box displays in the network diagram window and the Properties window for the task also displays. You can display different tasks in the Task Description dialog box by clicking the left and right facing arrows in the upper right corner of the dialog box.

2. Make your changes in the Task Description dialog box or Properties window.

   For details on task parameters, see “Task Parameters” on page 100.

Deleting Tasks

To delete a task:

1. On the task network diagram, select the task.

   When you select an item, a green box displays around the item.

2. Press the Del key on the keyboard. You can alternatively right-click the task in the tree view and select **Delete Task** from the menu that displays.

   The task is removed.

Variables

A list of variables associated with the model displays in the tree view below the Variables node and in the Variables window. You can display variable descriptions, and add, edit, and delete variables. You can also create watches to observe the changes in variables during model execution. For details on watches, see “Watches” on page 129.

Micro Saint Sharp recognizes global and local variables. Only global variables need to be defined. For details on the differences between the two variable types, see “Variable Scoping” on page 233.

For a list of system variables, see “System Variables” on page 121.

Displaying Defined Variables

To display defined variables:

In the tree view, click the Variables item to expand it.

A list of defined variables displays in the tree view and in the Variables window in the main Document window. The variable name, variable type, and initial value are listed.

Name	Type	Initial Value
AdjLength	int	0
Animator	MAAD.ActionView.Animator	0
BallLoc	int	0
Call	int	0
Clock	FloatingPoint	0.0
Distributions	MAAD.Simulator.Utilities.Dist	0
Down	int	1
DriveLength	FloatingPoint	70
Entity	Entity	0
FirstTeam	int	0
Gained	FloatingPoint	0
Model	MAAD.Simulator.Model	0
Passes	int	0

Displaying Variable Descriptions

To display a variable description:

1. In the tree view, click the Variables item to expand it.

2. Double-click the variable name. You can alternatively right-click the variable in the tree view and select **Edit Variable** from the menu that displays.

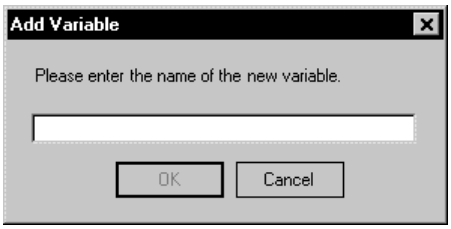
   The Variable Description dialog box displays in the main Document window and the associated Properties window for the variable displays.

Adding Variables

To add a variable:

1. In the tree view, right-click the Variables node or any existing variable and select **Add Variable** from the menu that displays.

   The Add Variable dialog box displays.



2. Type a variable name and click **OK**.

   Variable names must begin with a letter or underscore and can be followed by zero or more letters, numbers, and underscore characters (examples: `abc`, `_abc`, `_aa12)`. Variable names are case-sensitive. The OK button remains disabled if you enter an invalid name or the name of an already existing variable.

   The C# language reserves 76 words for its own use. Do not use these keywords as variable names. The keywords include the following:

   abstract, as, base, bool, break, byte, case, catch, char, checked, class, const, continue, decimal, default, delegate, do, double, else, enum, event, explicit, extern, false, finally, fixed, float, for, foreach, goto, if, implicit, in, int, interface, internal, is, lock, long, namespace, new, null, object, operator, out, override, params, private, protected, public, readonly, ref, return, sbyte, sealed, short, sizeof, stackalloc, static, string, struct, switch, this, throw, true, try, typeof, unit, ulong, unchecked, unsafe, ushort, using, virtual, void, and while.

The variable is added to the list of variables in the tree view and the Properties window displays.

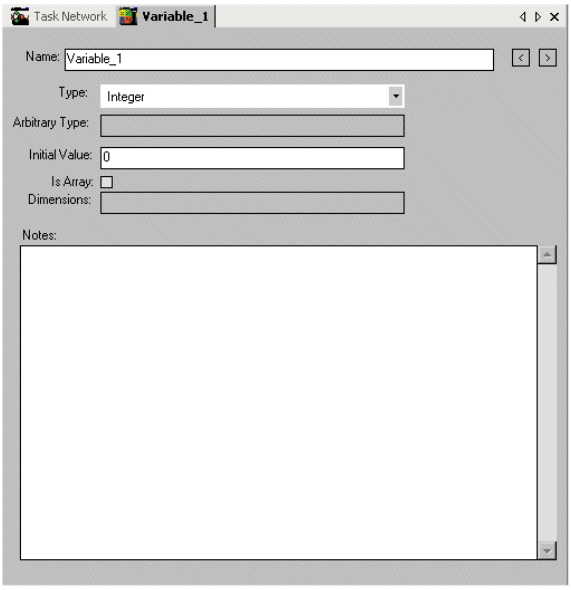
3. In the tree view, double-click the variable name to open the Variable Description dialog box.

4. Enter the variable parameters to define the variable.

   Parameters are described in the following section.

Variable Parameters

The parameters used to define a variable include the variable type, the initial value, whether the variable is an array, indexes for an array, and any variable notes.



Following are detailed descriptions of the variable parameters.

- **Name**

  Contains the name of the variable.

- **Type**

  Contains the variable type. Selections include integer, floating point, string, boolean, entity, object, and arbitrary.

  Integer variables are any whole number between approximately -2 billion and +2 billion (+2,147,483,648 and -2,147,483,647 inclusive). Variables that count discrete objects should be defined as integers.

  Floating Point variables, also called *doubles* or *real* numbers, are numbers that are not whole numbers. Floating point numbers can range between 5.0×10^{-324} to 1.7×10^{308} ; precision in Micro Saint Sharp is fifteen significant digits. Micro Saint Sharp displays floating point variables with four digits to the right of the decimal point. Variables that store clock values or measurements of other continuous quantities should be defined as floating point, because they can contain noninteger values.

  String variables are variables that consist of alphanumeric characters. You can use string variables in any expression. When you create a string variable, it does not need to be enclosed in quotes in the Initial Value text box. However, you must enclose the string variable in quotes if it is used in any other place, such as `string1 = "Micro Saint Sharp is great."`

  Boolean variables are variables that are true or false only. Boolean variables are often used as flags to indicate whether a condition is true or false.

  Entity variables point to a specific entity in the model.

  Object variables are variables that accept all types of values, from string to integer to floating point. These types of variables are mainly used with the Suspend and Resume functions that require a value of both a string and an integer. For example, if you wanted to Suspend task 3 with tag 3, then the Suspend function would require an array that holds both a string (for the task ID) and an integer (for the task tag). An object array would allow for this.

Arbitrary variables are variables that are of types not included in this list. Only advanced modelers should use this type of variable.

- **Initial Value**

  Contains the initial value for the variable. If the variable is an array, the initial value applies to all elements. You can also supply an initial value for a non-array variable or an element of an array by defining a scenario event that executes at time zero. Defining an initial value this way is equivalent to defining it in the Initial Value field. If you set the values in the Event Queue, you can make the value of the variables change at different times (for example, staffing shift changes).

- **Is Array**

  Indicates whether the variable is an array. An array is an ordered set of values or elements identified by a single name. Each array element is referred to by the array's name and the element's position in the array. The position is indicated by a set of index values enclosed in square brackets and separated by commas; index values can be constants or expressions. Arrays can contain integers or real numbers, but not both.

  Following is an example of an array named `Array1`. The dimensions of the array are 4 x 2. To reference the cell with the value 98, you would use `Array1[2,1]`.

```
           Col 0   Col 1

row 0      12      43

row 1      8       9

row 2      8       98

row 3      7       58
```

- **Dimensions**

  Contains the indices for the dimensions for an array. Type the upper index for each dimension, separated by a comma. The lower index for each dimension is zero. Micro Saint Sharp does not limit the number of dimensions you can use.

  For example, if you enter 30, 40, 50, 60, you are defining a four-dimensional array with indices of 0-29, 0-39, 0-49, and 0-59.

- **Notes**

  Contains a description of the variable. Entering notes is optional, but notes can be useful for future reference.

Editing Variables

To edit a variable:

1. In the tree view, click the Variables node to expand it and then double-click the variable to edit. You can alternatively right-click the variable in the tree view and select **Edit Variable** from the menu that displays.

   The Variable Description dialog box displays in the main Document window and the associated Properties window for the variable displays. You can display different variables in the Variable Description dialog box by clicking the left and right facing arrows in the upper right corner of the dialog box.

2. Make your changes.

   For details on variable parameters, see “Variable Parameters” on page 117.

Deleting Variables

To delete a variable:

1. In the tree view, click the Variable node to expand it and then click the variable to delete.

2. Press the Del key on the keyboard. You can alternatively right-click the variable and select **Delete Variable** from the menu that displays.

   The variable is deleted.

System Variables

Micro Saint Sharp uses the following system variables:

- `Clock` records elapsed time (in simulation time units) since the beginning of model execution. You can use `Clock` in any expression in a model, but use care in changing its value. To record the time at which an event occurred, set another variable equal to `Clock` as an effect of the event, for example, `FinishTime= Clock`. You can also use the variable `Clock` when the time controls what happens, for example:

  ```
  if (Clock >= 1200)
  {
    Operators= CrewSize;
  }
  ```

- `Distributions` consist of all the model distributions (used to generate random execution times). These are represented in the tree view below the Variables node by the single item `Distributions` (`MAAD.Simulator.Utilities.Distributions`).

- `Entity` consists of all the entity variables and functions. These are represented in the tree view below the Variables node by the single item `Entity (Entity)`.

- `Model` functions consist all the model functions, such as `Model.Start` or `Model.Pause`. These are represented in the tree view below the Variables node by the single item `Model(MAAD.Simulator.Model)`.

- Animator consists of all the animator objects. These are represented in the tree view below the Animator node by the single item `AnimatorNetwork` (Animator is an optional add-on component).

Entity Attributes

Entities are the conceptual objects that travel through a task network. The entity may represent a physical object, such as a part being built on a production line. It may also represent a person performing the tasks. Or, the entity may simply trace the sequence of tasks and not correspond to anything physical. You can create as many entity attributes as you need to track and record entity data.

As an example of an entity attribute you might add, consider the attribute product type. If you are simulating a manufacturing facility and products could be of type 1 or 2, you could add an entity attribute `ProductType` to the model. Then when the simulation sets the type of product for an entity, you could set the entity attribute (instead of setting an array variable with a `Tag` index).

A list of entity attributes associated with the model displays in the tree view below the Entity Attributes node and in the Entities Attributes window. You can display entity attribute descriptions, and add, edit, and delete entity attributes.

For a list of default entity attributes, see “Default Entity Attributes” on page 127.

Displaying Defined Entity Attributes

To display defined entity attributes:

In the tree view, click the Entity Attributes item to expand it.

A list of defined entity attributes displays in the tree view and in the Entity Attributes window. The attribute name, type, and initial value are listed.

Name	Type	Initial Value
Event	MAAD.Simulator.Utilities.EEventType	EEventType.BeginningEffect
Group	int	0
ID	string	0
qtime	FloatingPoint	0
Tag	int	0
Test	int	0
Time	FloatingPoint	0.0
Type	MAAD.Simulator.Utilities.EEntityType	EEntityType.StandardBegin
UniqueID	int	0
Test2	int	0

Displaying Entity Attribute Descriptions

To display an entity attribute description:

1. In the tree view, click the Entity Attributes item to expand it.

2. Double-click the entity name. You can alternatively right-click the entity in the tree view and select **Edit Entity Attribute** from the menu that displays.

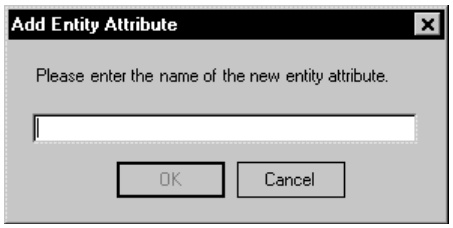
   The Entity Attribute Description dialog box displays in the main Document window and the associated Properties window for the attribute displays.

Adding Entity Attributes

To add an entity attribute:

1. In the tree view, right-click the Entity Attributes node or any existing entity attribute and select **Add Entity Attribute** from the menu that displays.

   The Add Entity Attribute dialog box displays.



2. Type an entity attribute name and click **OK**.

   Attribute names must begin with a letter or underscore and can be followed by zero or more letters, numbers, and underscore characters. Entity attribute names are case-sensitive. The OK button remains disabled if you enter an invalid name or the name of an already existing attribute.

   The attribute is added to the list of attributes in the tree view and the Properties window displays.

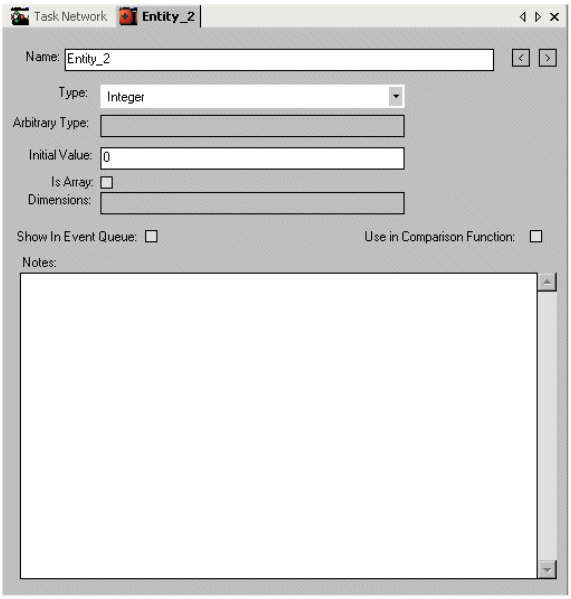
3. In the tree view, double-click the attribute name to open the Attribute Description dialog box.

4. Enter the attribute parameters to define the attribute.

   Parameters are described in the following section.

Entity Attribute Parameters

The parameters used to define an entity attribute include the entity type, the initial value, whether the entity is an array, indexes for an array, whether the entity attribute should display in the Event Queue, and whether the entity attribute is used in comparison functions.



Following are descriptions of the entity attribute parameters. For more details, see “Variable Parameters” on page 117.

- **Name**

  Contains the name of the entity attribute.

- **Type**

  Contains the entity attribute type.

- **Initial Value**

  Contains the initial value for the variable.

- **Is Array**

  Indicates whether the variable is an array.

- **Dimensions**

  Contains the indices for the dimensions for an array.

- **Show in Event Queue**

  Controls whether the entity displays in the event queue.

- **Use in Comparison Function**

  Controls whether Micro Saint Sharp compares entity attributes for equivalence. In some cases, you may want to merge multiple entities back together at a task. By default, any entities that have the same `Entity.Group`, `Entity.Tag`, and `Entity.ID` values are rejoined. If you select this parameter, Micro Saint Sharp also compares the entity attribute for equivalence.

- **Notes**

  Contains a description of the entity. Entering notes is optional, but notes can be useful for future reference.

Editing Entity Attributes

To edit an entity attribute:

1. In the tree view, click the Entity Attributes node to expand it and then double-click the entity to edit. You can alternatively right-click the entity in the tree view and select **Edit Entity Attribute** from the menu that displays.

   The Entity Attributes Description dialog box displays in the main Document window and the associated Properties window for the entity attribute displays. You can display different entity attributes in the Entity Attribute Description dialog box by clicking the left and right facing arrows in the upper right corner of the dialog box.

2. Make your changes.

   For details on entity attribute parameters, see “Entity Attribute Parameters” on page 125.

Deleting Entity Attributes

To delete an entity attribute:

1. In the tree view, click the Entity Attributes node to expand it and then click the entity attribute to delete.

2. Press the Del key on the keyboard. You can alternatively right-click the entity and select **Delete Entity Attribute** from the menu that displays.

   The entity attribute is deleted.

Default Entity Attributes

Micro Saint Sharp automatically defines the following attributes for each entity.

- `Entity.Duration` records the time each entity spends in the current task or queue. You can use `Entity.Duration` in any expression in the model. Any changes you make to the `Entity.Duration` attribute are ignored.

  Whenever an entity leaves a queue, Micro Saint Sharp sets `Entity.Duration` equal to the time the entity spends in the queue; you can record this value by setting another variable equal to `Entity.Duration` as a departing effect of the queue. For example, to record the value of duration in the departing effect, you could use the following expression:

  ```
  QueueWait[Entity.Tag] = Entity.Duration;
  ```

■ Entity Attributes

Whenever an entity starts a task, Micro Saint Sharp sets `Entity.Duration` equal to the task execution time for the entity; you can record this value in the launch effect of the task. For example, to record the value of duration in the launch effect, you could use the following example:

```
TaskTime[Entity.Tag] = Entity.Duration;
```

`Entity.Duration` is only valid in the Departing Effect for queues and the Launch Effect field for tasks.

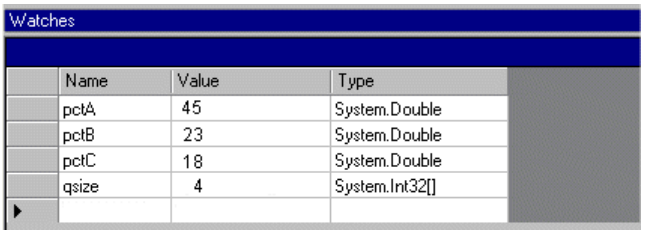
- `Entity.Group` is identical to the `Entity.Tag` attribute but is used to control groups of tasks. As with the `Entity.Tag` attribute, the `Entity.Group` value is carried through the simulation.
- `Entity.ID` records the task number in which the entity is in.
- `Entity.Tag` records the identity of each entity when you have multiple entities traveling through a network. Each entity is assigned a default value of zero when it is generated, but you can assign a new `Entity.Tag` value to an entity at any point. Once an entity has a `Entity.Tag` value, the value stays with that entity through the remainder of model execution or until you change it. When you use `Entity.Tag` in an expression, Micro Saint Sharp interprets it as the number of the entity that is currently being processed.
- `Entity.Time` tracks the closest scheduled event time for the entity.
- `Entity.Type` records the event type for which entity is scheduled. Event types can be one-time or repeating.
- `Entity.UniqueID` maintains the unique ID for the entity as defined by Micro Saint Sharp.

✓ Note:
When you use an entity attribute in an expression, precede the attribute with the word `Entity`, such as `Entity.Tag++;` or `Entity.Group = Entity.Tag +199;`

Watches

You can monitor the values of variables as a model executes. Viewing the values of variables in the model can be an excellent tool for debugging.

To display the values of a variable, you create a watch for the variable. The values of the variable then automatically display in the Watches window during model execution. The variable name, value, and type of variable display.



Adding Watches

To add a watch:

1. In the tree view, right-click the Watches node or any existing watch and select **Add Watch** from the menu that displays.

2. The Add Watch dialog box displays.

3. Type the name of the variable and click **OK**.

   You can only create a watch for a defined variable. If you enter the name of an undefined variable, the watch is ignored.

   You can enter specific array values for a variable name. For example, if you have an array named `ArrayName` in the task network, you can specify a particular item in the array such as `ArrayName[5]`. For the purpose of a watch, the indexer must be a numerical value and not a variable (for example, `ArrayName[Variable]` is not valid).

   The watch is added to the list of watches in the tree view and the Properties window displays.

4. In the Notes box in the Properties window, enter any notes regarding the watch.

   The next time you execute the model, the values of the variable display in the Watches window.

Editing Watches

To edit a watch:

1. In the tree view, click the Watches node to expand it and then click the watch to edit.

   The Properties window for the watch displays.

2. Make your changes.

Deleting Watches

To delete a watch:

1. In the tree view, click the Watches node to expand it and then click the watch to delete.

2. Press the Del key on the keyboard. You can alternatively right-click the watch and select **Delete Watch** from the menu that displays.

   The watch is deleted.

Functions

Custom or user-defined functions are useful when you need to perform the same calculation or procedure several times in a model. Instead of retyping the necessary expressions every place you need them, you can first create the function once for a model. Then, wherever you need the calculation or procedure to be performed in the model, you simply type the name of the function followed by parentheses ( ).

Because you can assign more than one expression to a function, you can use functions to perform procedures. For example, suppose you want to assign values to several variables based on the state of the simulation at various points in the model. You could define a function called `SystemChange( )`. Whenever the function is executed, Micro Saint Sharp executes all of the expressions in the function and returns the value of the last expression.

A list of functions associated with the model displays in the tree view below the Functions node. Functions also display in the associated Functions window.

Displaying Defined Functions

To display defined functions:

In the tree view, click the Functions item to expand it.

A list of defined functions displays in the tree view and in the Functions window in the main Document window. The function name, type, parameters, and code defining the function are listed.

Name	Type	Parameters	Code
ChangeTime	void		tmpclock = (1+ Math.Floor (Clock/900))*900-Clock;
MoveBall	void		int curTeam;
SwitchTeams	void		YdsToGo=10;
UpdateScore	void		int Qtr;

Displaying Function Descriptions

To display a function description:

1. In the tree view, click the Functions item to expand it.

2. Double-click the function name. You can alternatively right-click the function in the tree view and select **Edit Function** from the menu that displays.

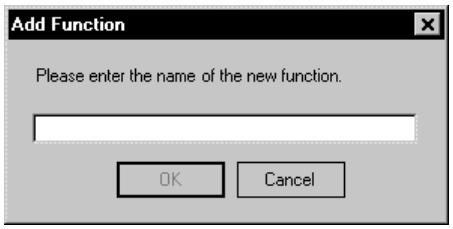
   The Function Description dialog box displays in the main Document window and the associated Properties window for the function displays.

Adding Functions

To add a function:

1. In the tree view, right-click the Functions node or any existing function and select **Add Function** from the menu that displays.

   The Add Function dialog box displays.



2. Type a function name and click **OK**.

   Function names must begin with a letter or underscore and can be followed by zero or more letters, numbers, and underscore characters. Function names are case-sensitive. The OK button remains disabled if you enter an invalid name or the name of an already existing function.

   The C# language reserves 76 words for its own use. Do not use any of these keywords as function names. For a list of reserved names, see page 116.

   The function is added to the list of functions in the tree view and the Function Properties window displays.

3. In the tree view, double-click the function name to open the Function Description dialog box.

4. Enter the parameters defining the function.

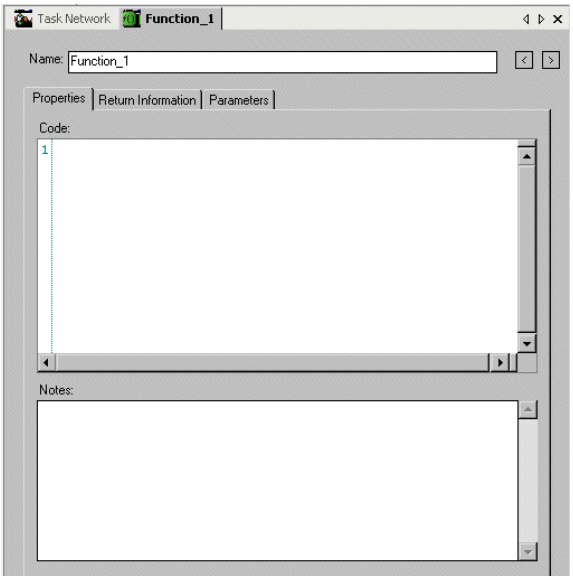
   Parameters are described in the following section.

Function Parameters

Function parameters are included in three tabs, the Properties tab, the Return Information tab, and the Parameters tab.

Properties Tab

The Properties tab contains the expressions defining the function and a section for notes regarding the function.



Following is a detailed description of the parameters in the Properties tab.

- **Code**

  Contains the expressions defining the function. To return a value from the function, you must include a return statement in the function definition. The variable type for the return value is specified in the Return Information tab. For details on return statements, see “Return Statements” on page 236.

- **Notes**

  Contains any information describing the function. Entering notes is optional, but notes can be useful for future reference.

Return Information Tab

The Return Information tab contains the information regarding the value that is returned from the function. A function is not required to return a value.

Following are detailed descriptions of the parameters in the Return Information tab.

- **Name**

  Contains the function name.

- **Type**

  Contains the variable type for the return value. Selections include integer, floating, string, boolean, entity, arbitrary, object, and void. Void types do not return a value. For descriptions, see “Variable Types” on page 118.

- **Arbitrary Type**

  Contains the return type when the value is not one of the default return types included with Micro Saint Sharp, such as a function that returns a Socket object. Type the return type in this text box.

- **Is Array**

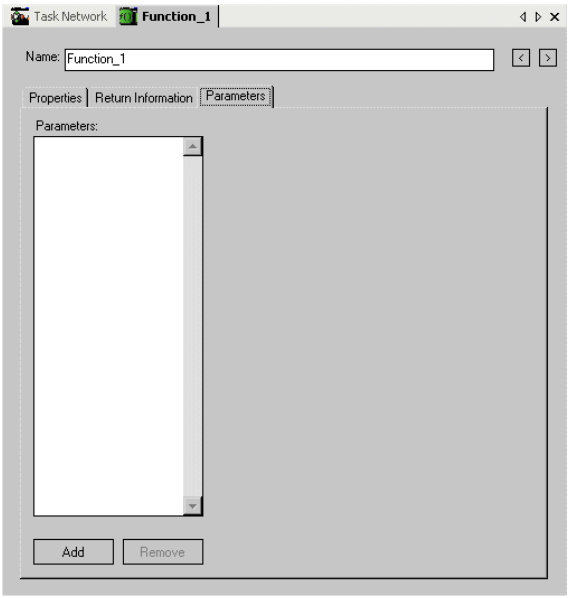
  Indicates whether the return value is an array.

- **Dimensions**

  Contains the indexes for the return value when it is an array. Type the upper index for each dimension, separated by a comma. By default the lower index for each dimension is zero. For example, if you enter 30, 40, 50, 60, you are defining a four-dimensional array with indices of 0-29, 0-39, 0-49, and 0-59. There is no limit on the number of dimensions you can use.

Parameters Tab

The Parameters tab contains optional variables that a function can assign to a local variable inside the function. Function parameters are temporary variables that are sent into a function.



Following are detailed descriptions of the fields in the Function Parameters tab.

- **Parameters**

  Contains a list of optional values that the function can call. For example, you could add a parameter called `a` to the parameter list for a function. When the function code is executed, the parameter `a` is available, such as `return a*10;`

  To add a variable, click the **Add** button and enter the Name, Type, Array information, and Arbitrary type. To remove a variable, select it and click the **Remove** button. For details on the variable parameters, see "Variable Parameters" on page 117.

Editing Functions

To edit a function:

1. In the tree view, click the Functions node to expand it and then double-click the function to edit. You can alternatively right-click the function in the tree view and select **Edit Function** from the menu that displays.

   The Function Description dialog box displays in the main Document window and the associated Properties window for the function displays. You can display different functions in the Function Description dialog box by clicking the left and right facing arrows in the upper right corner of the dialog box.

2. Make your changes.

   For details on function parameters, see “Function Parameters” on page 134.

Deleting Functions

To delete a function:

1. In the tree view, click the Functions node to expand it and then click the function to delete.

2. Press the Del key on the keyboard. You can alternatively right-click the function and select **Delete Function** from the menu that displays.

   The function is deleted.

Scenario Events

Scenario events provide a way for you to cause certain events to occur at specific times during model execution. These can be one-time events or they can repeat at regular intervals. Scenario events are often used to change variable values and thereby change the state of the model. For example, you could have a variable called `Temperature` that would increase at one-hour intervals during the day, and decrease at night. You could then make the time for a specific task such as warming up an engine dependent upon the current temperature.

Scenario events assign values to variables independent of when an entity begins or ends a task or enters or departs a queue. Scenario events supersede what is happening on the task network.

To view the execution of scenario events, open the Event Queue window. Scenario events only display in the Event Queue window during model execution.

A list of scenario events associated with the model displays in the tree view below the Scenario Events node and in the Scenario Events window. You can display scenario event descriptions, and add, edit, and delete scenario events.

Displaying Defined Scenario Events

To display defined scenario events:

In the tree view, click the Scenario Events item to expand it.

A list of defined events displays in the tree view and in the Scenario Event window in the main Document window. The Scenario Event window displays the event name, the start time, whether the event is repeating, the repeating interval, whether a repeating event should stop before the end of the simulation and the stop time, and the expression describing the event.

Name	Start Time	Repeating	Repeating Interval	Stop	Stop Time	Code
Initialize	0	☐	0	☐	0	ProcAssem = 1;
Event 2	0	☑	1	☐	0	DispAssem = 1;

Displaying Scenario Event Descriptions

To display a scenario event description:

1. In the tree view, click the Scenario Events item to expand it.

2. Double-click the event name. You can alternatively right-click the event in the tree view and select **Edit Scenario Event** from the menu that displays.

   The Scenario Event Description dialog box displays in the main Document window and the associated Properties window for the event displays.

Adding Scenario Events

To add a scenario event:

1. In the tree view, right-click the Scenario Events node or any existing event and select **Add Scenario Event** from the menu that displays.

   The scenario event is added to the list of events in the tree view and the Scenario Events Properties window displays. The event is assigned the name `ScenarioEventN` where `N` is the next available integer.

2. In the tree view, double-click the event name to open the Scenario Events Description dialog box.

3. Enter the parameters defining the event.

   Parameters are described in the following section.

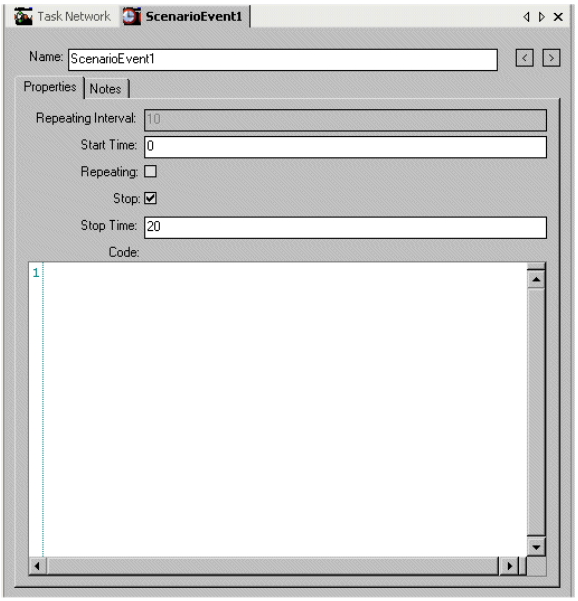
Scenario Event Parameters

Scenario Event parameters are included in two tabs, the Properties tab and the Notes tab.

The Scenario Event name displays at the top of each tab. You can change the name from the default. Event names must begin with a letter or underscore and can be followed by zero or more letters, numbers, and underscore characters. Spaces are not allowed. Event names are case-sensitive.

Properties Tab

The Properties tab contains start time for the event, whether the event is repeating, the stop time for the event, and the expressions defining the scheduled event.



Following are detailed descriptions of the parameters in the Properties tab.

- **Repeating Interval**

  Specifies the time interval at which an event should repeat.

- **Start Time**

  Contains the time for the event to occur.

- **Repeating check box**

  Indicates whether the event should repeat.

- **Stop check box**

  Indicates that a repeating event should stop before the end of the simulation.

- **Stop time**

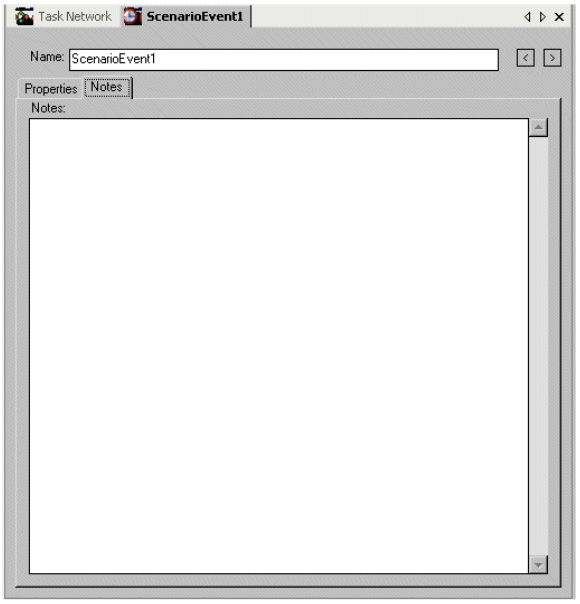
  Contains the time for the event to stop repeating.

- **Code**

  Contains the expressions that defines the scenario event. These expressions occur at the times you specify.

Notes Tab

The Notes tab contains notes regarding the event. Entering notes is optional, but notes can be useful for future reference.



Editing Scenario Events

To edit a scenario event:

1. In the tree view, click the Scenario Events node to expand it and then double-click the scenario event to edit. You can alternatively right-click the event in the tree view and select **Edit Scenario Event** from the menu that displays.

   The Scenario Event Description dialog box displays in the main Document window and the associated Properties window for the scenario event displays. You can display different scenario events in the Scenario Description dialog box by clicking the left and right facing arrows in the upper right corner of the dialog box.

2. Make your changes.

   For details on event parameters, see “Scenario Event Parameters” on page 141.

Deleting Scenario Events

To delete a scenario event:

1. In the tree view, click the Scenario Events node to expand it and then click the event to delete.

2. Press the Del key on the keyboard. You can alternatively right-click the event and select **Delete Scenario Event** from the menu that displays.

   The scenario event is deleted.

Snapshots

Snapshots record the values of particular variables at specified points during model execution. The Snapshots window displays the snapshots that are defined to collect data for the current model.

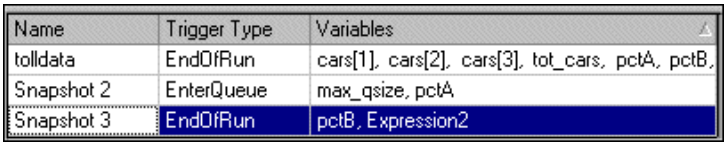
A list of snapshots associated with the model displays in the tree view below the Snapshot node and in the Snapshots window. You can display snapshot descriptions, and add, edit, and delete snapshots.

Displaying Defined Snapshots

To display defined snapshots:

In the tree view, click the Snapshots item to expand it.

A list of defined snapshots displays in the tree view and in the Snapshots window in the main Document window. The name, trigger type, and variables collected are listed.



Name	Trigger Type	Variables
tolldata	EndOfRun	cars[1], cars[2], cars[3], tot_cars, pctA, pctB,
Snapshot 2	EnterQueue	max_qsize, pctA
Snapshot 3	EndOfRun	pctB, Expression2

Displaying Snapshot Descriptions

To display a snapshot description:

1. In the tree view, click the Snapshots item to expand it.

2. Double-click the snapshot name. You can alternatively right-click the snapshot in the tree view and select **Edit Snapshot** from the menu that displays.

   The Snapshot Description dialog box displays in the main Document window and the associated Properties window for the event displays.

Adding Snapshots

To add a snapshot:

1. In the tree view, right-click the Snapshots node or any existing snapshot and select **Add Snapshot** from the menu that displays.

   The snapshot is added to the list of snapshots in the tree view and the Snapshot Properties windows displays. The snapshot is assigned the name `SnapshotN` where `N` is the next available integer.

2. In the tree view, double-click the snapshot name to open the Snapshot Description dialog box.

3. Enter the parameters defining the snapshot.

   Parameters are described in the following section.

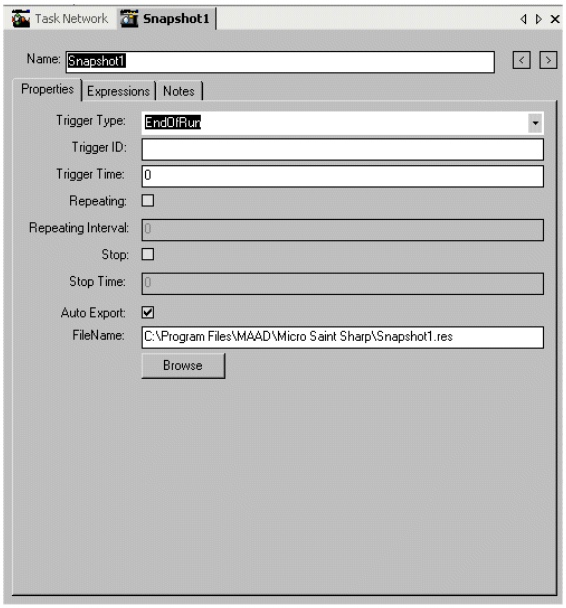
Snapshot Parameters

Snapshot parameters are included in three tabs, the Properties tab, the Expressions tab, and the Notes tab.

The Snapshot name displays at the top of each tab. You can change the name from the default. Snapshot names must begin with a letter or underscore and can be followed by zero or more letters, numbers, and underscore characters. Spaces are not allowed. Snapshot names are case-sensitive.

Properties Tab

The Properties tab contains the trigger type, trigger ID, trigger time, repeating intervals, and stop time for the snapshot. The tab additionally contains settings to automatically export the snapshot results and indicate the export location.



Following are detailed descriptions of the parameters in the Properties tab.

- **Trigger Type**

  Contains the condition to trigger the snapshot. Selections available from the drop-down list box include the following:

  Begin Task
  Triggers the snapshot at the beginning of the task.

  End Task
  Triggers the snapshot at the end of the task.

Clock
Triggers the snapshot at a specific time. The system variable `Clock` is automatically included in every snapshot description that you add, but you can delete it.

Enter Queue
Triggers the snapshot as the queue is entered.

Depart Queue
Triggers the snapshot as the queue is departed.

End of Run
Triggers the snapshot at the end of the run.

- **Trigger ID**

  For triggering tasks or queues, this field contains the number of the task or queue.

- **Trigger Time**

  Contains the starting time if the snapshot has a clock trigger.

- **Repeating**

  Select this check box for the snapshot to be repeating. When you select this check box, the Repeat interval field is enabled; it contains the repeat interval.

- **Stop check box**

  Select this check box if the snapshot has a clock trigger.

- **Stop time**

  Contains the stop time if the snapshot has a clock trigger.

- **Auto Export**

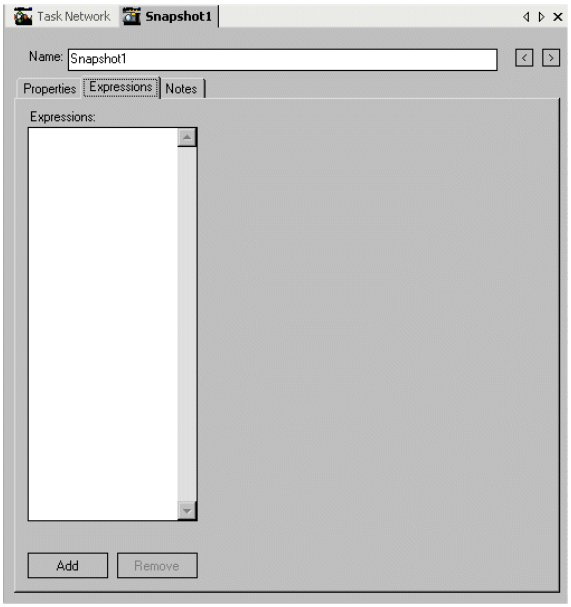
  Select this check box to automatically export the snapshot file.

- **File Name**

  Enter or select the file name and location for the snapshot file.

Expressions Tab

The Expressions tab contains the variables or expressions that you want stored.



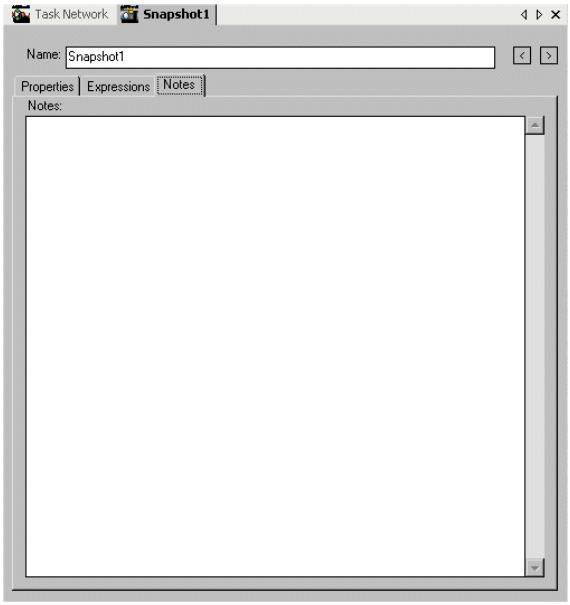
Following are detailed descriptions of the parameters in the Expressions tab.

- **Expressions**

  Contains the expressions for which you want Micro Saint Sharp to record values. To add an expression, click the **Add** button and enter the expression in the Expression text box. To remove an expression, select the expression and click the **Remove** button. After you have defined snapshots, you must also enable their collection in the Execution Settings dialog box. For details, see “Running the Model” on page 157.

Notes Tab

The Notes tab contains any notes about the snapshot. Entering notes is optional, but notes can be useful for clarity and future reference.



Editing Snapshots

To edit a snapshot:

1. In the tree view, click the Snapshots node to expand it and then double-click the snapshot to edit. You can alternatively right-click the snapshot in the tree view and select **Edit Snapshot** from the menu that displays.

   The Snapshots Description dialog box displays in the main Document window and the associated Properties window for the snapshot displays. You can display different snapshots in the Snapshot Description dialog box by clicking the left and right facing arrows in the upper right corner of the dialog box.

2. Make your changes.

   For details on snapshot parameters, see “Snapshot Parameters” on page 146.

Deleting Snapshots

To delete a snapshot:

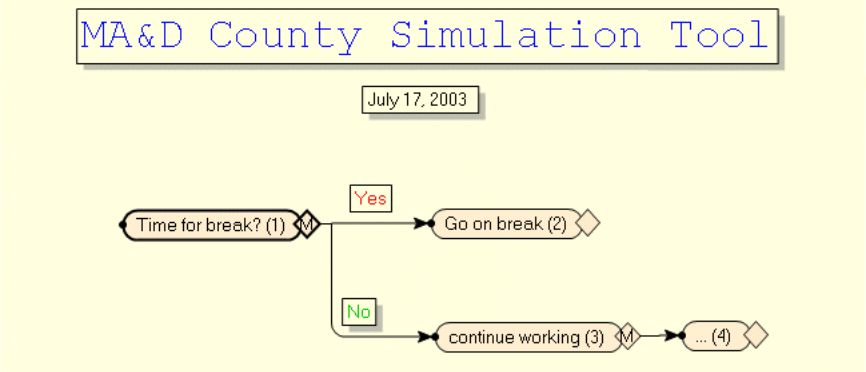
1. In the tree view, click the Snapshots node to expand it and then click the snapshot to delete.

2. Press the Del key on the keyboard. You can alternatively right-click the snapshot and select **Delete Snapshot** from the menu that displays.

   The snapshot is deleted.

Comments

You can use comments to add clarification to the task network diagram and to display values of variables on the task network diagram as the model runs. For example, you could place the name, description, and last revision date of the model as a title on the task network diagram. You could also place comments describing the different path logic in places where you had a decision node with multiple path routing. The following illustration shows the use of comments in the title and in path routing.



You can position comments in the same way as any other network object. A list of comments associated with the model displays in the tree view below the Task Network node. You can display comment descriptions, and add, edit, and delete comments.

Displaying Comment Descriptions

To display a comment description:

On the task network diagram, double-click the comment.

The Comment Description dialog box displays in the main Document window and the associated Properties window for the comment displays.

Adding Comments

To add a comment:

1. From the Task Network palette, drag the Comment tool to the correct place on the task network diagram. You can alternatively right-click in the task network and select **Add Comment** from the menu that displays.

2. Double-click the comment box.

   The Comment dialog box displays.

3. Enter the comment parameters.

   Parameters are defined in the following section.

Comment Parameters

Comment parameters display at the top of the Comment Description dialog box. Additional parameters are contained in the Variables and the Appearance and Notes tab.

Parameters at the top of the window include the following:

- **Name**

  Contains the name of the comment. Enter the text that you want to display on the task network diagram. The Comment box automatically resizes to contain the text.

- **ID**

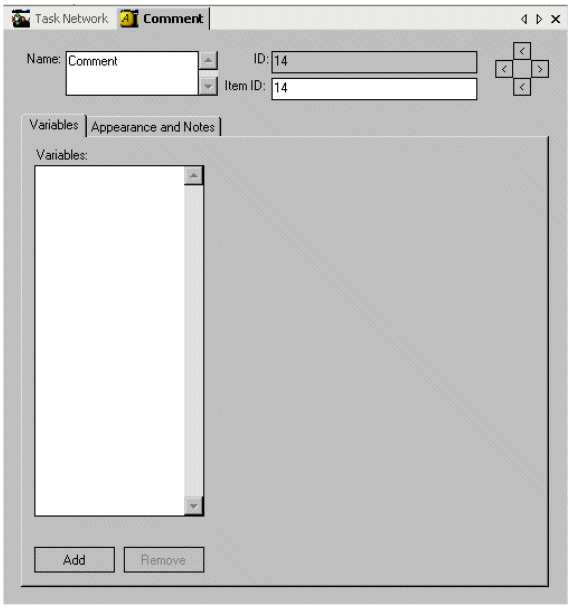
  Contains the Micro Saint Sharp-defined ID number for the comment. Micro Saint Sharp automatically assigns the next available number when a new comment is created. The ID reflects the network hierarchy. To change the last digit of the Micro Saint Sharp ID, enter a new ID in the Item ID box.

- **Item ID**

  Contains the user-defined ID for the task.

Variables Tab

The Variables tab contains a list of variables to display in the Comment box on the task network diagram during model execution. As examples, you might want to display the rate of entities arriving at a task and the size of a queue as the model executes.



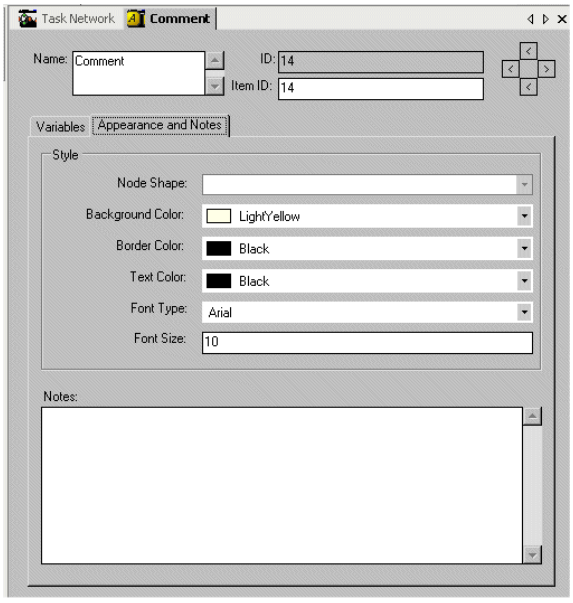
Following is a detailed description of the parameter in the Variables tab.

- **Variables**

  Contains a list of variables to display during model execution in the Comments box. To add a variable, click the **Add** button and enter the expression or variable name. To remove a variable, select it and click the **Remove** button.

Appearance and Notes Tab

The Appearance and Notes tab contains the factors controlling the style of the Comment box and text within the box and contains any notes regarding the comment.



Following are detailed descriptions of the parameters in the Appearance and Notes tab.

- **Style**

  This section contains the items that control the physical appearance of the Comment box. For details on these items, see “Network Parameters” on page 92.

- **Notes**

  Contains any notes about the comment. Entering notes is optional, but notes can be useful for clarity and future reference.

Editing Comments

To edit a comment:

1. On the task network diagram, double-click the comment. You can alternatively right-click the comment in the tree view and select **Edit Comment** from the menu that displays.

   The Comment Description dialog box displays in the main Document window and the Properties window for the comment also displays. You can display different comments in the Comment Description dialog box by clicking the left and right facing arrows in the upper right corner of the dialog box.

2. Make your changes.

   For details on comment parameters, see “Comment Parameters” on page 153.

Deleting Comments

To delete a comment:

1. On the task network diagram, select the Comment box.

2. Press the Del key on the keyboard. You can alternatively right-click the comment in the tree view and select **Delete Comment** from the menu that displays.

   The comment is deleted.

8

Running the Model

Once you have created a model and defined the model parameters, you are ready to run the model. Several settings control model execution. You can set the number of runs for the model, set the random seed, indicate the start ID, control the type of output for snapshot files, enable Line by Line debugging, enable data collection, set the execution speed factor, and enter initialization code and compiler instructions.

Execution options allow you to you start, resume, pause, or halt model execution; step through model execution one event at a time; and change the speed of execution. Other options include updating the event queue, charts, and Comment variables to display during model execution and showing network animation.

This chapter also contains a discussion of potential sources of errors in logic and syntax, and describes how to use the syntax checker provided by Micro Saint Sharp.

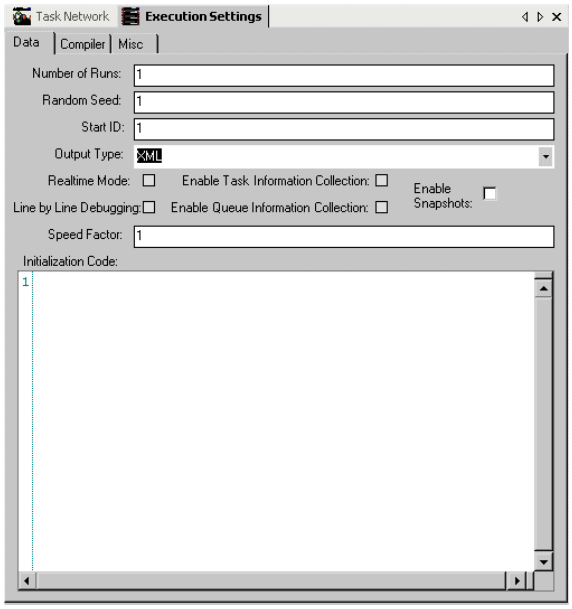
Displaying Execution Settings

Execution Settings are contained in three tabs and control the number of runs for the model, the random seed and start ID, type of output, whether Line by Line debugging is enabled, the type of information collected, the speed factor, the initialization code, compiler instructions, and miscellaneous information.

To display execution settings:

In the tree view, double-click the Execution Settings item.

The Execution Settings dialog box and the associated Properties window display.



Editing Execution Settings

Before running a model, check the execution settings to make sure they are as you want them. The execution settings are saved as part of the model settings. You should also run the Syntax checker before you execute the model. For details, see “Checking for Syntax Errors” on page 172.

To edit execution settings:

1. In the tree view, double-click the Execution Settings item.

   The Execution Settings dialog box and the associated Properties window display.

2. Edit the execution parameters.

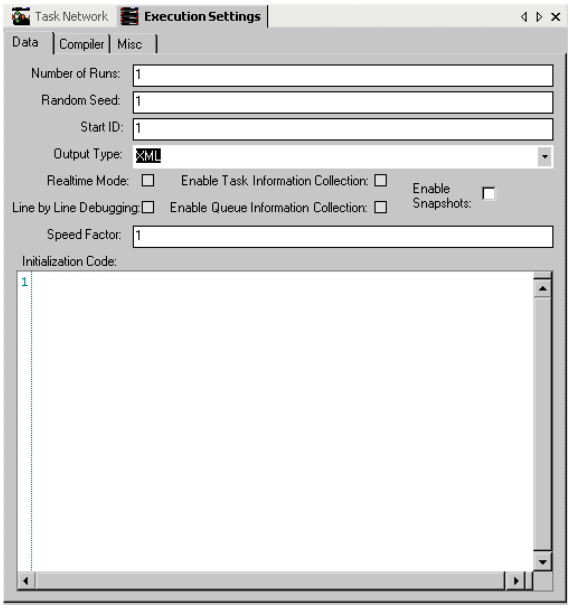
   Parameters are described in the following section.

Execution Setting Parameters

Execution Settings parameters are included in three tabs, the Data, Compiler, and Misc tabs.

Data Tab

The Data tab contains settings for the number of runs, the random seed, the start ID, the type of output, whether real time mode is used, whether data collection is enabled, whether line-by-line debugging is used, the speed factor, and the initialization code.



Following is a detailed description of the execution parameters.

- **Number of Runs**

  Contains the number of times to run the model. If the number is greater than one, the execution times, paths of execution, and other probabilistic events are different for each run. If you run the model multiple times with the Task Information, Queue Information, and Snapshot options enabled, the results files contain data from all of the consecutive runs.

- **Random Seed**

  Contains the number used to generate random numbers for calculating task execution times and taking probabilistic paths through the network. The random number seed is a number between approximately -2 billion and +2 billion (inclusive). The same seed always produces the same set of random numbers in the same order, and thus the same results.

- **Start ID**

  Contains the number of the task or network marked as the starting place for the model simulation. You can set the start node on the task network diagram (see "Designating the Start Node" on page 77) or you can enter the task or network number in the Start ID box.

- **Output Type**

  Controls the format of the output of snapshot data collection. To store the results in XML format, select XML; results files have the extension `.xml`. To store the results in ASCII format, select Tab Delimited; results files have the extension `.res`.

- **Realtime Mode**

  Controls whether the model runs in actual time or in simulated time. To execute the model according to real time required for tasks, select the RealTime Mode check box. Otherwise, the execution runs in simulated time.

- **Line by Line Debugging**

  Controls whether line by line debugging is used. When enabled, the Line Debugger window displays and increases the resolution for debugging a model simulation. When the Line Debugger window is open and you step through the model simulation, execution occurs one line of code at a time. When the Line Debugger window is not open and you step through the simulation, execution occurs one effect at a time rather than one line of code at a time.

♦ **Enable Task Information Collection**

Controls whether task information is collected; select the check box to enable task information. Task information includes a record of how long each task requires to execute. You must also mark each task individually for task data collection in the Main tab in the Task Description dialog box (see "Main Tab" on page 102); by default, Micro Saint Sharp marks each task for data collection.

The following information is collected:

- Task number
- Task name
- Minimum task time
- Maximum task time
- Mean task time
- Standard deviation (calculated using the nonbiased or n-1 method)
- Number of times the task was performed ("task hits")

♦ **Enable Queue Information Collection**

Controls whether queue information is collected; select the check box to enable queue information collection. You must also mark each task individually for queue data collection in the Queue tab in the Task Description dialog box (see "Queue Tab" on page 109); by default, Micro Saint Sharp marks each queue for data collection. Queue information includes the following variables, collected each time a queue is encountered.

- `Clock`
- `Entity.Tag`
- `Run`
- `Length`
- `Wait`

- **Enable Snapshots**

  Controls whether Micro Saint Sharp stores values of specified variables in a snapshot file when triggered by specified events; select the check box to collect snapshots. You create snapshots in the Snapshots Description dialog box (see "Adding Snapshots" on page 146).

Note:
Collecting data can slow down system performance, particularly when you have several snapshots. You may want to disable all data collection until you are specifically interested in collecting the data.

- **Enable Layout Change**

  Controls whether the window configuration automatically changes from the Edit to Execution configuration during model execution; select the check box to enable automatic change. For additional details, see "Window Configurations" on page 39.

- **Speed Factor**

  Controls the speed of the simulation for realtime mode simulation. A 2 speed factor means the speed is twice as fast as real time and a .5 speed factor means the speed is half as fast as real time. Valid values are any real number.

Note:
The speed factor only functions when the model is run in Realtime mode (see page 161).

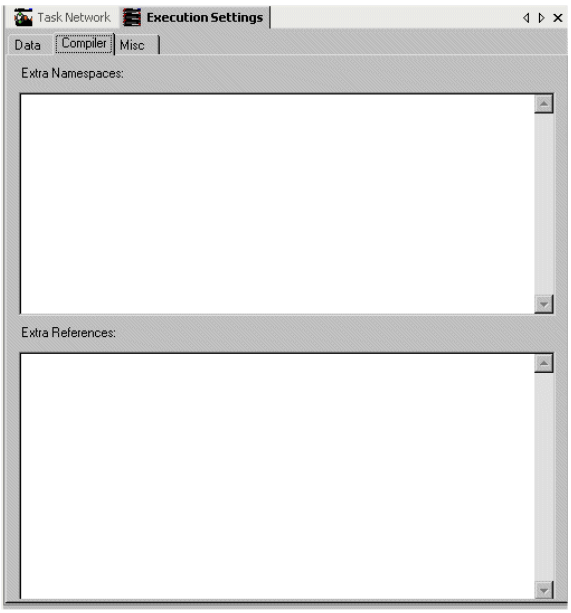
The simulation speed setting on the Micro Saint Sharp toolbar works in conjunction with the Speed factor set in the Execution Preferences dialog box. For example, if you enter a speed factor of 2 in the Execution Preferences dialog box, and select 200% from the toolbar, the net speed factor is four.

♦ **Initialization Code**

Contains any expressions to run before the model starts. For example, if you are running a model that connects to an application on the internet, you would enter the code that executes to connect to the external application.

Compiler Tab

The Compiler tab contains settings for advanced users who want to use other `.dlls` in their Micro Saint Sharp code.



♦ **Extra Namespaces**

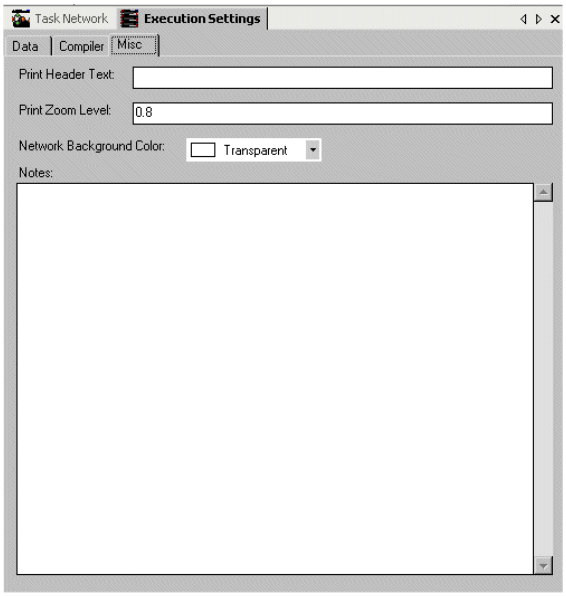
Contains the namespaces to use from the loaded dlls, such as `System.Drawing`, `System.Drawing.Design`, and `System.Windows.Forms`.

♦ **Extra References**

Contains the names of the `.dll` files, such as `System.Drawing.dll` and `System.Windows.Forms.dll`.

Misc Tab

The Misc tab contains settings for printing the task network diagram. To print the network diagram and set other print properties, see “Printing the Task Network Diagram” on page 86.



- **Print Header Text**

  Contains the text to print at the top of the network diagram.

- **Print Zoom Level**

  Contains the zoom level for the network diagram.

- **Network Background Color**

  Specifies the background color for the network diagram.

- **Notes**

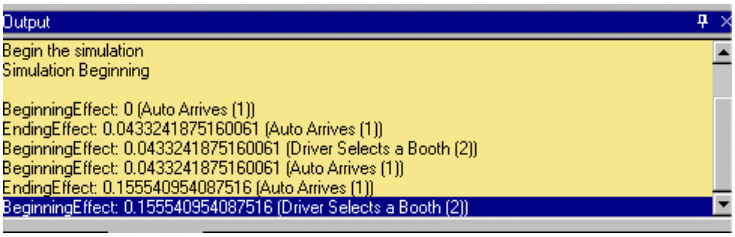
  Contains any notes regarding the print setup.

Viewing Trace Data During Simulation

The Output window displays the actions that occur when you run the model and is called the trace data. The following information displays:

- Clock values for beginning effects
- Clock values for ending effects
- Clock values for scenario events
- Syntax errors
- Whether the plugins (such as Animator `.dlls`) are successfully loaded when Micro Saint Sharp is started
- When a run or simulation begins and ends
- Application errors

You can save the results in the Output window to a file, save the results to the Windows clipboard, and clear the data in the window.


Output
Begin the simulation
Simulation Beginning
BeginningEffect: 0 (Auto Arrives (1))
EndingEffect: 0.0433241875160061 (Auto Arrives (1))
BeginningEffect: 0.0433241875160061 (Driver Selects a Booth (2))
BeginningEffect: 0.0433241875160061 (Auto Arrives (1))
EndingEffect: 0.155540954087516 (Auto Arrives (1))
BeginningEffect: 0.155540954087516 (Driver Selects a Booth (2))

To display trace data in the Output window:

From the **Utilities** menu, select **Display Trace** so that a check mark displays adjacent to Display Trace.

The next time you run the model simulation, trace information displays in the Output window.

To disable trace data in the Output window:

From the **Utilities** menu, select **Display Trace** so that the check mark adjacent to Display Trace clears.

The next time you run the model simulation, trace information does not display in the Output window. The only information that displays is the simulation clock value, run number, and duration of simulation.

✓ Note:
When you disable the display of trace data, you cannot enable network animation, charts, event queue, or comment variables to display.

Clearing the Output Window

You can clear or remove the results in the Output window so that the information in the window only applies to the current simulation. If you want to save the results for a single simulation, be sure to clear the Output window before the simulation or you will save the results from all simulations.

To clear the Output window:

From the **Edit** menu, select **Clear Output**.

The Output window clears.

Saving the Results in the Output Window to a File

You can save the results in the Output window to a text file.

To save the results in the Output window to a file:

1. From the **Edit** menu, select **Save Output** to **File**.

   The Save As dialog box displays. The default location for the file is the current Output directory and the default name is `Output.txt`.

2. Navigate to the folder where you want to save the file and enter the file name.

3. Click **Save**.

   The output file is saved and you can open it in any ASCII text editor.

Saving the Results in the Output Window to the Clipboard

To save the results in the Output window to the clipboard:

1. From the **Edit** menu, select **Copy Output to Clipboard**.

   The output file is copied to the Windows clipboard.

2. Open the text editor you want to use to view the results and paste the information that is on the clipboard.

Viewing Scheduled Events During Simulation

The Event Queue window displays the list of events as they occur during model execution. These include the execution of tasks and any defined scheduled events. The Group, ID, Tag, Time, and Type of event display.

Beginning effects are highlighted in navy, ending effects are in green, suspended beginning effects are in blue, and suspended ending effects are in lime. Evaluate queue is in fuchsia, leaving queue is in purple, and waiting without queue is in red. And finally, scenario events are in yellow and snapshots are in maroon.

Event	Group	ID	Tag	Time	Type
EndingEffect	0	assign	1	4.43622446	Task
EndingEffect	0	patient	1	24.5122013	Task
ScenarioEvent	0	ScenarioEvent10	0	480	ScenarioEvent
ScenarioEvent	0	ScenarioEvent11	0	840	ScenarioEvent

To view the execution of scenario events, open the Event Queue window. Scenario events only display in the Event Queue window during model execution.

To update scheduled events during model execution:

From the **Utilities** menu, select **Event Queue** so that a check mark displays adjacent to the label Event Queue.

The next time you run the model simulation, the scheduled events display in the Event Queue.

✓ Note:
You must enable the display of trace data before you can enable the display of the scheduled events in the Event Queue.

To disable the update of scheduled events during model execution:

From the **Utilities** menu, select **Event Queue** so that the check mark adjacent to the label Event Queue clears.

The next time you run the model simulation, the scheduled events do not display in the Event Queue.

Updating Charts During Simulation

When you execute a Micro Saint Sharp model, you can generate charts showing variable information. These charts display in the Chart window and can be updated during the model simulation. For information on adding charts to a model, see “Variable Data Files” on page 183. Disabling the display of charts speeds model execution.

✓ Note:
You must enable the display of trace data before you can enable the display of charts (see page 166).

To update charts during model execution:

From the **Utilities** menu, select **Display Charts** so that a check mark displays adjacent to the label Display Charts.

The next time you run the model simulation, the Chart window displays.

To disable the update of charts during model execution:

From the **Utilities** menu, select **Display Charts** so that the check mark adjacent to the label Display Charts clears.

The next time you run the model simulation, the Chart window does not display.

Updating Comment Variables During Simulation

You can select variables to display on the task network diagram during model execution in Comment boxes. For details on selecting the variables to display, see “Adding Comments” on page 152. Once you select the variables, you can enable or disable the update during model execution. To maximize execution speed, disable the update of Comment variables to display.

Note:
You must enable the display of trace data before you can enable the update of comment variables (see page 166).

To update Comment variables during model execution:

From the **Utilities** menu, select **Update Comment Variables** so that a check mark displays adjacent to the label Update Comment Variables.

The next time you run the model simulation, any variables selected for display in Comments dynamically update as the model executes.

To disable the update of Comment variables during model execution:

From the **Utilities** menu, select **Update Comment Variables** so that the check mark adjacent to the label Update Comment Variables clears.

The next time you run the model simulation, any variables selected for display in Comments do not update.

Viewing Network Animation During Simulation

When you execute a Micro Saint Sharp model, you can view the model execution as it occurs using network animation. When you enable network animation, you can watch entities move through the network diagram. When an entity is in a task, the task highlights in a blue color. The number of entities in a task or queue are indicated by numbers above that display above the task or queue. Disabling network animation speeds model execution.

Note:
You must enable the display of trace data before you can enable the display of network animation (see page 166).

To enable the display of network animation:

From the **Utilities** menu, select **Network Animation** so that a check mark displays adjacent to the label Network Animation.

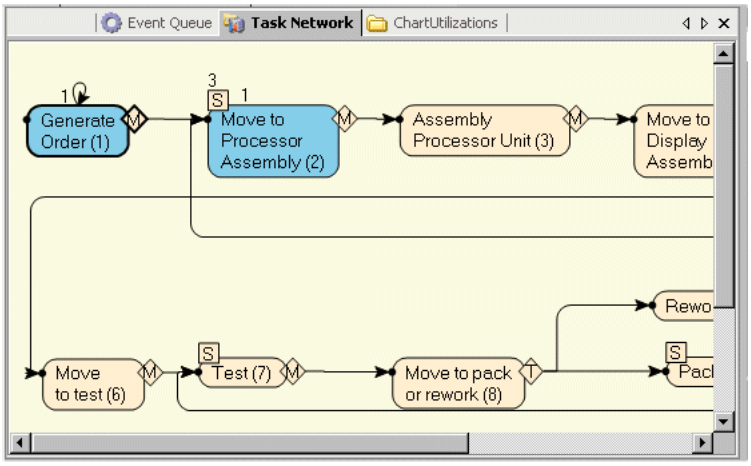
The next time you run the model simulation, network animation displays in the Task Network window.

To disable the display of network animation:

From the **Utilities** menu, select **Network Animation** so that the check mark adjacent to the label Network Animation clears.

The next time you run the model simulation, network animation does not display in the Task Network window.

The following illustration shows an example of a step in the network animation. At the specific time in model execution, one entity is in task 1 and one entity is in task 2. Three entities are in the queue before task 2.



Checking for Syntax Errors

Micro Saint Sharp has a feature that checks a model for syntax errors. You can run the syntax check at any time and the entire model is checked. Syntax errors display in the Output window, so you must first open the window.

To check for syntax errors:

1. From the **View** menu, select **Windows**, and then select **Output**.

   The Output window displays.

2. From the **Execution** menu, select **Check for Errors**.

   Any errors that are detected display in the Output window.

3. To open the dialog box where the error occurs, double-click the error.

Note:
The syntax checker does not check for logic errors.

Micro Saint Sharp generally indicates the nature of the problem in the error message, such as “unrecognized word” for a misspelled or undeclared variable or function, or “semicolon expected at end” for a missing semi-colon.

Common problems include typographical errors in an expression, omitting a semi-colon, forgetting to define a variable, or using the wrong case for a variable name. Because Micro Saint Sharp distinguishes between upper and lower case letters, name, NAME, and Name are considered to be different and unique variables.

Checking for Logic Errors

Micro Saint Sharp displays error messages for some of the more common logic errors such as an array index that is out of bounds or division by zero. Micro Saint Sharp has several different built-in tools that you can use to help you find where the logic errors are in a model.

- Watches and Event Queue windows. If you leave these windows open as a model executes, you can watch the variables change in value and view the events as they occur. Change the execution speed to a slower speed if the model executes too quickly for you to observe the changes in the values.

- Comment boxes can display the values of variables at any location in the task network.
- The Output window displays the actions that occur when you run the model. Any syntax errors are indicated and you can use them for debugging purposes.
- The Task Information data file can also be used to help find errors in logic. The file contains a record of the beginning time, ending time, and `Enity.Tag` numbers for each task executed during the model run.

Using the Line Debugger Window

The Line Debugger window is a separate window that you can use to increase the resolution for debugging a model simulation.

When the Line Debugger window is open and you step through model simulation, execution occurs one line of code at a time. When the Line Debugger window is not open and you step through the simulation, execution occurs one effect at a time rather than one line of code at a time. You can only enable line debugging between model runs.

To display the Line Debugger window:

In the Execution Settings dialog box, select the Line by Line Debugging check box.

Controlling Execution Speed

In addition to starting and stopping model execution, Micro Saint Sharp provides commands to pause, resume, and step through execution. You can also set the speed for model execution.

Starting and Resuming Model Execution

Before executing the model, check the execution preferences to make sure the model is set to run the way you want it.

To start or resume model execution:

From the **Execution** menu, select **Begin Simulation** or press Ctrl-G.

The model executes or resumes execution.

Pausing Model Execution

You can temporarily pause or stop a model execution. When model execution is paused, you can make changes to the model attributes such as the value of variables.

To pause model execution:

1. From the **Execution** menu, select **Pause Simulation** or press Ctrl-P.

   The model pauses executing.

2. When you are ready to resume model execution, select **Begin Simulation** from the **Execution** menu.

Stepping Through Model Execution

You can also step through model execution one event at a time. This method of execution is useful for debugging a model. This command is available only when the model has not yet begun execution, or when model execution is paused.

To step through model execution:

1. From the **Execution** menu, select **Step Simulation** or press Ctrl-T.

   Micro Saint Sharp executes the next event and pauses.

2. Select **Step Simulation** again.

   Micro Saint Sharp executes the next event and pauses.

3. To resume continuous execution, select **Begin Simulation** from the **Execution** menu.

Stopping Execution

You can stop the current execution of a model at any point. When you halt model execution, *it cannot be resumed*. The run is terminated and must start over.

To stop model execution:

From the **Execution** menu, select **Halt Simulation** or press Ctrl-H.

The current execution stops. To stop model execution temporarily and then resume it, use the Pause command. If you click **Begin Simulation** after you have halted an execution, a new run is started.

Setting the Execution Speed for RealTime Mode Simulation

You can control the speed of realtime model simulation from the Micro Saint Sharp toolbar (for details on Realtime mode simulation, see page 161). Execution speeds are 25%, 50%, 100%, 200%, 400%, and 800%; each execution speed is twice as fast as the speed below it.

The speed settings on the Micro Saint Sharp toolbar work in conjunction with the Speed factor set in the Execution Preferences dialog box. For example, if you select 200% from the toolbar and enter a speed factor of 2 in the Execution Preferences dialog box, the net speed factor is four.

To maximize execution speed, disable network animation, disable the display of trace data, and disable the update of charts, scheduled events, and comment variables.

✓ Note:
You can change the speed during execution.

To set the execution speed:

1. From the **Utilities** menu, select **Simulation Speed**.

2. Select the execution speed from the list that displays.

■ Controlling Execution Speed

9

Analyzing Model Data

When you run a model, Micro Saint Sharp produces data regarding the model run based on parameters that you set up in the Execution Settings dialog box. You can collect trace, task, queue, and snapshot data and export it to external applications for analysis.

In addition to these data sets, you can collect the values of the variables during model execution. Micro Saint Sharp provides twenty different charts that you can use for data analysis. Charts are listed in the tree view and are saved with each model. You can display a list of all defined charts, display the definition for an individual chart, and add, edit, and delete charts. You can also format charts, copy charts, paste charts, print charts, and save charts as an image file.

Trace Data Files

The trace data file contains a record of the beginning and ending time of each task as the model executes. Trace data displays in the Output window as a model executes. The following information displays:

- `Clock` value
- Clock times for beginning effects, ending effects, and scenario events
- Application errors
- Whether plugins were loaded successfully
- Start and end of a simulation

To collect trace data, you must enable the display of trace data. For details, see “Viewing Trace Data During Simulation” on page 166.

To analyze trace data, save the trace results in the Output window and then open it in the application you want to use for analysis. For details on saving the trace results, see “Saving the Results in the Output Window to a File” on page 167.

Task Data Files

The task data file contains a record of how long each task requires to execute. The following information displays for each task:

- Task number
- Task name
- Minimum task time
- Maximum task time
- Mean task time

- Standard deviation (calculated using the nonbiased or n-1 method)
- Number of times the task was performed (“task hits”)

To collect Task Data, you must enable Task Data collection for each individual task. Additionally, you must also enable Task Information Collection in the Execution Settings dialog box. For details, see “Execution Setting Parameters” on page 159.

To analyze the trace data, export the data and then open it in the software application you want to use for analysis. For details on exporting data, see “Exporting Model Information” on page 63.

Queue Data Files

The queue data file contains a record of the queue information. The following information displays for each queue:

- Queue ID – ID number for the queue.
- Name – name of the queue.
- Number Of Queue Hits – number of times an entity entered the queue.
- Minimum Queue Length – minimum amount of time an entity spent in the queue.
- Maximum Queue Length – maximum amount of time an entity spent in the queue.
- Queue Length Mean – mean amount of time an entity spent in the queue.
- Queue Length Standard Deviation – standard deviation for the mean amount of time an entity spent in the queue.

- Minimum Queue Wait – minimum amount of time an entity spent waiting in the queue before being allowed to execute the task.
- Maximum Queue Wait – maximum amount of time an entity spent waiting in the queue before being allowed to execute the task.
- Queue Wait Mean – mean amount of time an entity spent waiting in the queue before being allowed to execute the task.
- Queue Wait Standard Deviation – standard deviation for the mean amount of time that an entity spent waiting in the queue before being allowed to execute the task.

To collect queue data, you must enable Queue Information Collection in the Execution Settings dialog box. For details, see “Execution Setting Parameters” on page 159.

To analyze the queue data, export the data and then open it in the software application you want to use for analysis. For details on exporting, “Exporting Model Information” on page 63.

Snapshot Data Files

The snapshot data file contains snapshot data collected during model execution. You specify the data to collect in a snapshot in the Snapshot Description dialog box. For details, see “Adding Snapshots” on page 146.

To collect snapshot data, you must enable Snapshots in the Execution Settings dialog box. For details, see “Execution Setting Parameters” on page 159.

Variable Data Files

In addition to task, trace, queue, and snapshot data file sets, you can collect information regarding the values of the variables during model execution. You can plot these values within Micro Saint Sharp on any of twenty different types of charts.

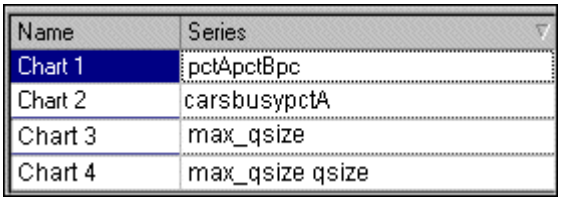
Charts are listed in the tree view and are saved with each model. You can display a list of all defined charts, display the definition for an individual chart, and add, edit, and delete charts. You can also format charts, print charts, and save charts as an image file.

Displaying Defined Charts

To display defined charts:

In the tree view, double-click the Charts item to expand it.

A list of existing charts displays in the tree view and in the Charts window in the main Document window. The Charts window displays the chart name and the variables included in the chart.



Displaying Chart Descriptions

To display a chart description:

1. In the tree view, click the Charts item to expand it.
2. Double-click the chart name.

   The Chart Description dialog box displays in the Chart window and the associated Properties window for the chart displays.

Adding Charts

To add a chart:

1. In the tree view, right-click the Chart node or any existing chart and select **Add Chart** from the menu that displays.

   Micro Saint Sharp automatically numbers the chart and adds it to the tree view below the Charts node.
2. To display the chart description, double-click the chart name in the tree view.

   The Chart Description dialog box displays. The dialog box contains two tabs: Properties and Series List.
3. Enter the chart parameters.

   Parameters are defined in the following section.
4. Format the chart.

   For details, see “Formatting Charts” on page 199.

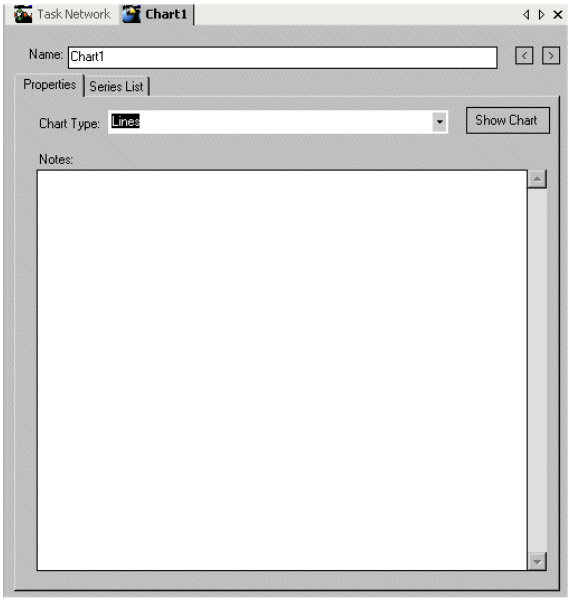
Chart Parameters

The chart name displays at the top of the Chart Description dialog box. Additional parameters are contained on the Properties and Series List tab. Following is a description of the chart parameters.

Use a name that is descriptive of the chart, for example, Line Graph for PercentA.

Properties Tab

The Chart Properties tab contains the chart type and any notes regarding the chart.



Following are detailed descriptions of the parameters in the Chart Properties tab.

- **Chart Type**

  Displays the selected chart type. For a description of charts, see “Types of Charts” on page 189.

- **Notes**

  Contains any notes regarding the chart.

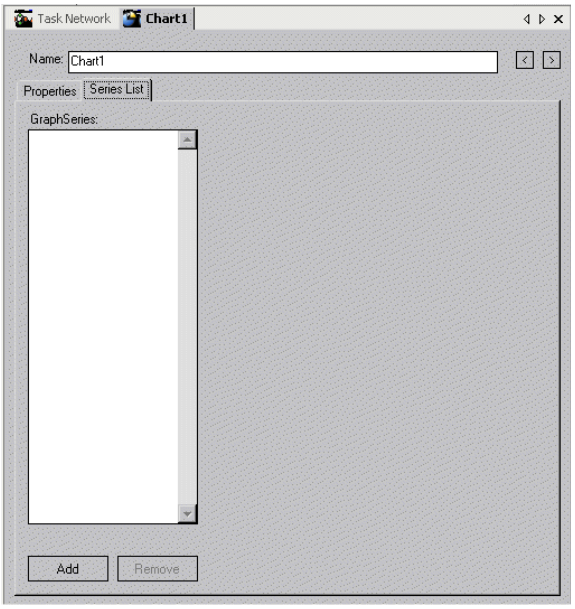
- **Show Chart**

  Click this button to display an example of the selected chart type.

Series List Tab

The Series List tab contains the data series (variables) to be plotted and the display characteristic for the data points. Each data series can have a unique color or pattern.

Note:
You can also format the chart from the Chart display; several additional parameters are available. For details, see “Formatting Charts” on page 199.



Following are detailed descriptions of the parameters in the Series List tab.

- **GraphSeries**

  Contains the list of the series to be plotted. A series can be a variable or expression. Each variable or expression that you enter constitutes a series.

  To add a series, click the **Add** button and enter the Name, X, Y, Z, Color, and Shape. To remove a series, select it and click the **Remove** button. You can include more than one graph series in a chart, except for pie charts.

✓ Note**:**
By default, the Clock variable is plotted on the Y axis. To change the variable that displays on the Y axis, click the series in the tree view. In the Properties window, enter the value to use on the Y axis in the Y text box.

- **Name**

  Contains the name of the series selected in the GraphSeries list.

- **X**

  Sets the variable to be plotted on the X axis.

- **Y**

  Sets the variable to be plotted on the Y axis.

- **Z**

  Sets the variable to be plotted on the Z axis.

- **Color**

  Controls the data points color for the selected series.

- **Shape**

  Controls the data points shape for the selected series.

Editing Charts

To edit a chart:

1. In the tree view, click the Charts node to expand it and then double-click the chart to edit.

   The Chart Description dialog box displays in the main Document diagram window and the associated Properties window for the chart displays. You can display different charts in the Chart Description dialog box by clicking the left and right facing arrows in the upper right corner of the dialog box.

2. Make your changes.

   For details on chart parameters, see “Chart Parameters” on page 185.

Deleting Charts

To delete a chart:

1. In the tree view, click the Charts node to expand it and then click the chart to delete.

2. Press the Del key on the keyboard. You can alternatively right-click the mouse and select **Remove Chart** from the menu that displays.

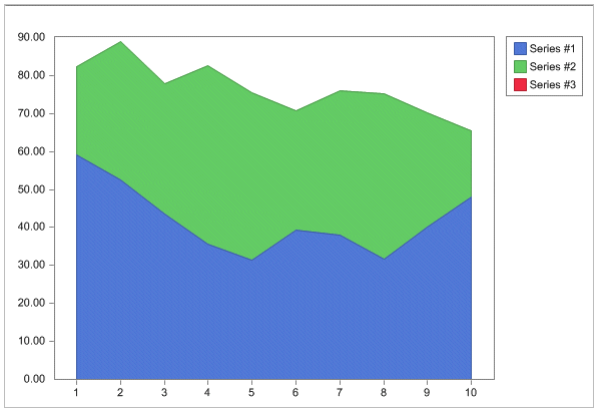
   The chart is deleted.

Types of Charts

Micro Saint Sharp provides twenty types of charts for you to use as data analysis tools. Charts are integrated from tools provided by Chart FX for .NET. For additional examples of the charts provided by Chart FX for .NET, see www.softwarefx.com.

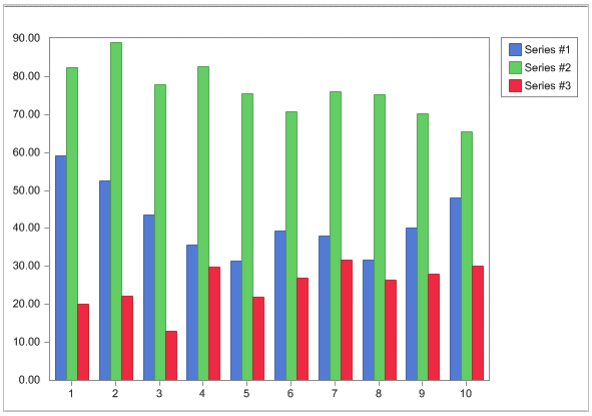
Area Charts

Area charts emphasize the degree of change over time and show the relationship of parts to a whole. By displaying the sum of the plotted values, area charts show the relationship of the parts to the whole. Area charts are useful for noting trends.



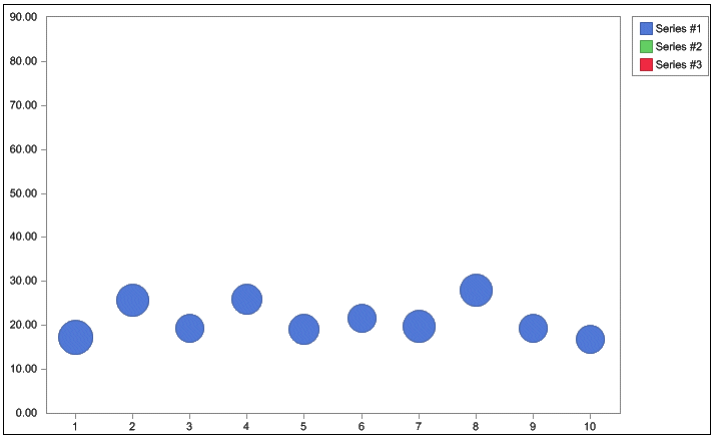
Bar Charts

Bar charts compare individual items. Categories are organized vertically and values are organized horizontally to place more emphasis on comparing values and less emphasis on time.



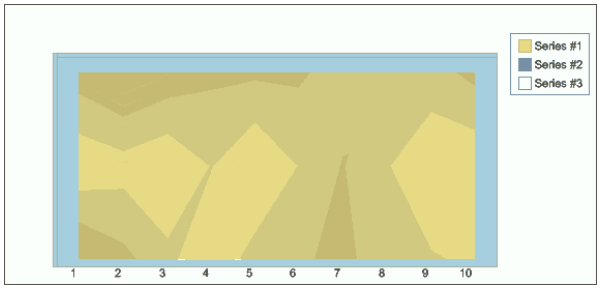
Bubble Charts

Bubble charts are a variation of X-Y line graphs where the data points are replaced by bubbles. The bubbles provide a way to display a third variable. Either the diameter or area of each bubble is proportional to the value it represents. Bubble charts are used instead of line charts when the data has a third dimension that needs to be shown on the chart.



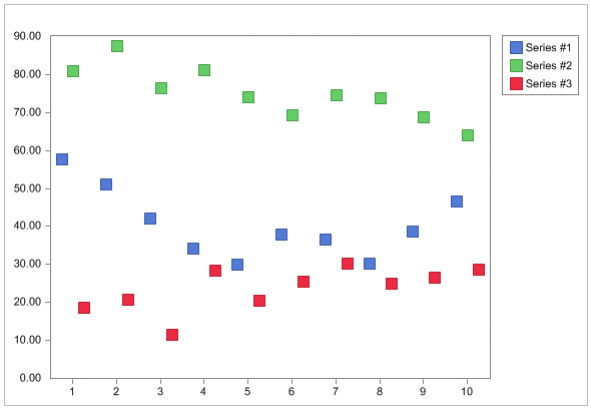
Contour Charts

Contour charts are useful for representing large numbers of values visually against each other using a set scale. Contour charts are also useful for noting trends.



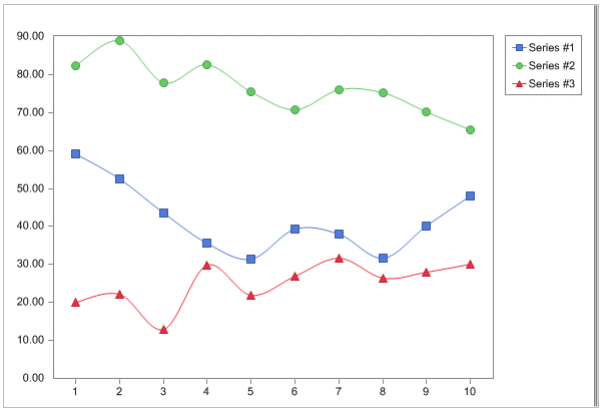
Cube Charts

Cube charts are best used when comparing a set or series of values to the same scale or unit of comparison or over time. Cube charts are useful for displaying multidimensional, hierarchical data.



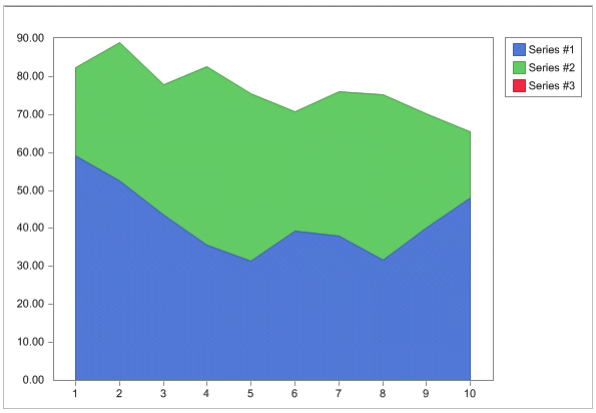
Curve Charts

Curve charts emphasize the degree of change over time and show the relationship of the parts to the whole. By displaying the sum of the plotted values, curve charts show the relationship of parts to the whole. Curve charts are useful for noting trends.



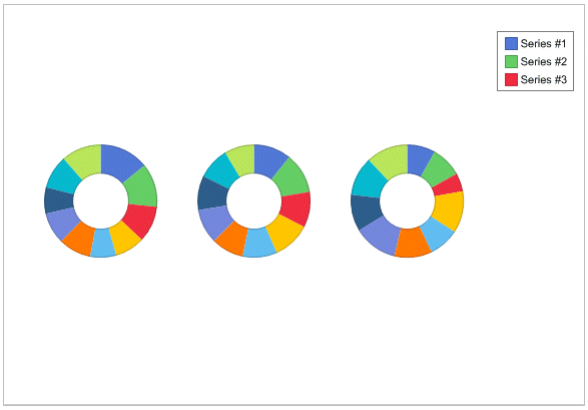
Curve Area Charts

Curve Area charts emphasize the degree of change over time and show the relationship of parts to a whole. By displaying the sum of the plotted values, curve area charts show the relationship of parts to the whole. Curve Area charts are useful for noting trends.



Doughnut Charts

Doughnut charts show the relationship of the parts to a whole, but can contain more than one data series. Each ring of the doughnut chart represents a data series. Doughnut charts are useful to compare sectional values of a whole to each other and the whole.



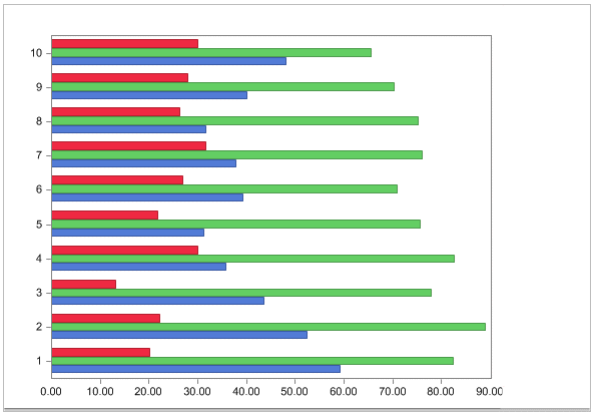
Financial Charts

Micro Saint Sharp provides the following financial charts:

- Open HiLowClose charts. Generally used to display statistics of a security over a measured amount of time.
- Candlestick charts. Generally used to display statistics of a security over a measured amount of time. This type of chart consists of a series of boxes (both open and closed) with a small spike above and below the box. Each box with its associated spikes typically represent one day, week, or month of trading.
- HiLowClose charts. Generally used to display statistics of a security over a measured amount of time. This is one of the most widely used types of charts in commodity futures trading. It consists of a series of vertical lines with two horizontal lines on the left and right of each horizontal line. Each vertical line can typically represent one day, week, or month of trading.

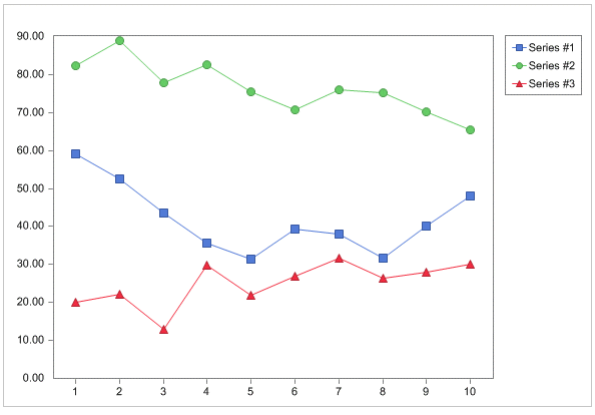
Gantt Charts

Gantt charts are horizontal bar charts that are best used to compare a set or series of values to the same scale or unit of comparison or over time. Gantt charts are frequently used in project management and provide a graphical illustration of a schedule.



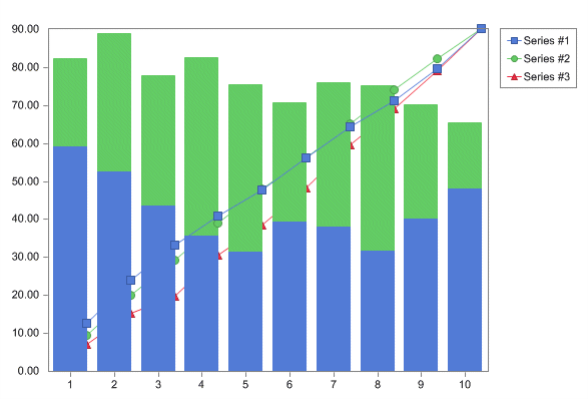
Line Charts

Line charts are well suited for comparing data or series of values over a recorded period of time. Line charts are useful for noting trends.



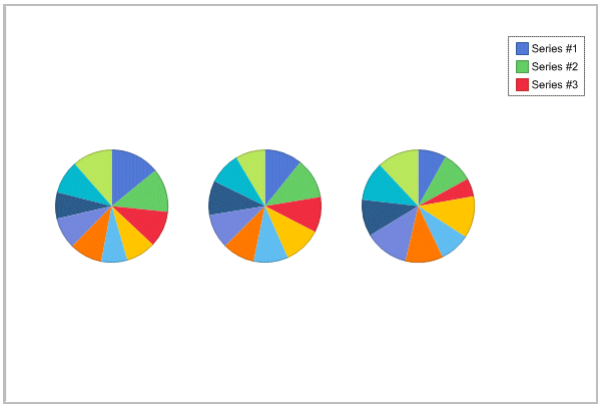
Pareto Charts

Pareto charts are a specialized version of a histogram that rank the categories in the chart from most frequent to least frequent. Pareto charts are often used to show the most common causes of defects or common causes of out-of-control conditions.



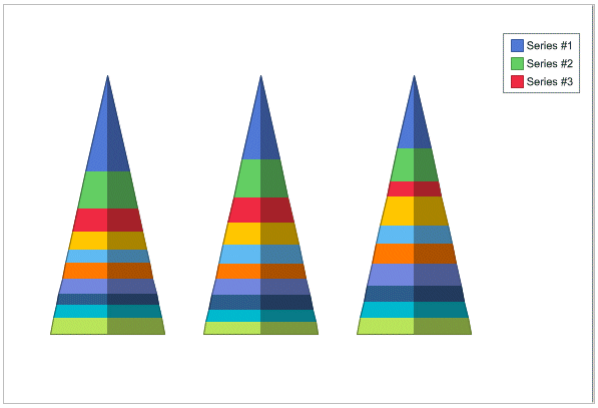
Pie Charts

Pie charts show the relationship of items that make up a data series to the sum of the items. Pie charts always show only one data series and are useful to emphasize a significant element. Pie charts are best for showing percentages of a whole.



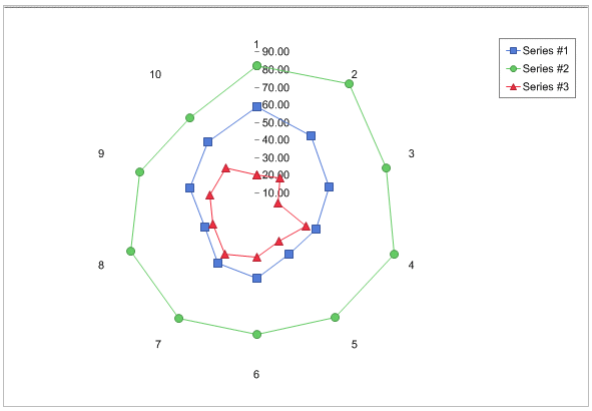
Pyramid Charts

Pyramid charts can be useful in comparing sectional values of a whole to each other and to the whole. Pyramid charts are an effective way to dramatically display bar charts. Pyramids are columns with pyramid shape.



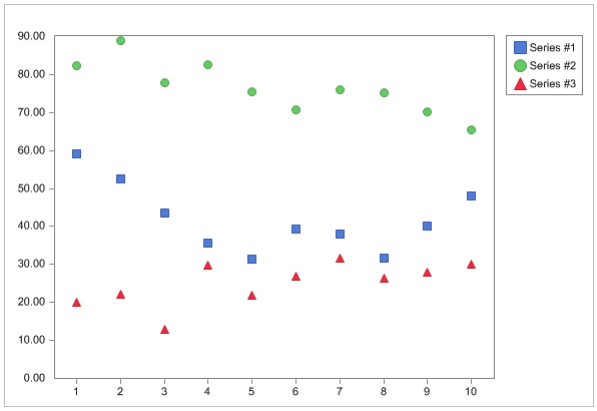
Radar Charts

Radar charts are useful for defining performance and identifying strengths and weaknesses. In a radar chart, each category has its own value axis radiating from the center point. Lines connect all the values in the same series. Radar charts display the relational differences between values. Radar charts are sometimes referred to as star or spider charts.



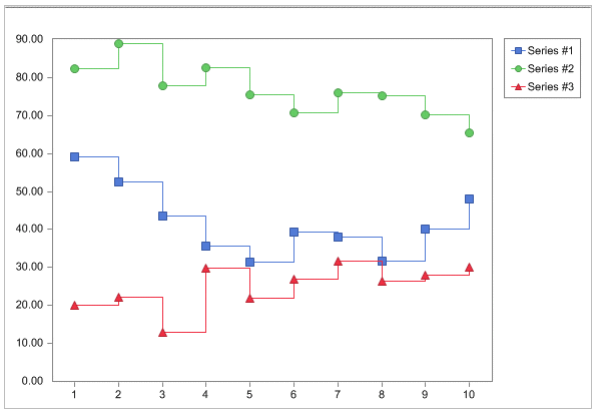
Scatter Charts

Scatter charts show the degree of relationship between values in several series. Scatter charts display observations as a chart showing the values of two variables, one dependent and one independent.



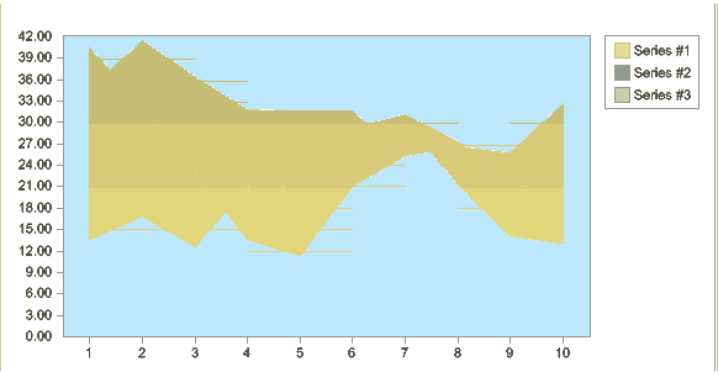
Step Charts

Step charts are well suited for comparing data or series of values over a recorded period of time. They are useful for noting trends.



Surface Charts

Surface charts are useful for comparing large numbers of values visually using a set scale. Surface charts are useful when you want to find optimum combinations between two sets of data. Colors and patterns indicate areas that are in the same range of values, similar to a topographic map.



Formatting Charts

The charting algorithm used in Micro Saint Sharp is integrated from tools provided by Chart FX for .NET. Chart FX provides a wide variety of options for formatting charts. Formatting options include the General, Series, and Y Axis tab in the Chart Properties dialog box, formatting the labels on the axes, adding or removing the Data Editor, and positioning and formatting the legend.

Many of these options are also available on the Chart toolbar. To display the Chart toolbar, right-click anywhere in the chart and select **Toolbar**.

To format a chart:

1. Right-click anywhere on the data series and select **Properties** from the menu that displays.

   The Properties dialog box displays with the Series tab in front. Three tabs are available: General, Series, and Y Axis.

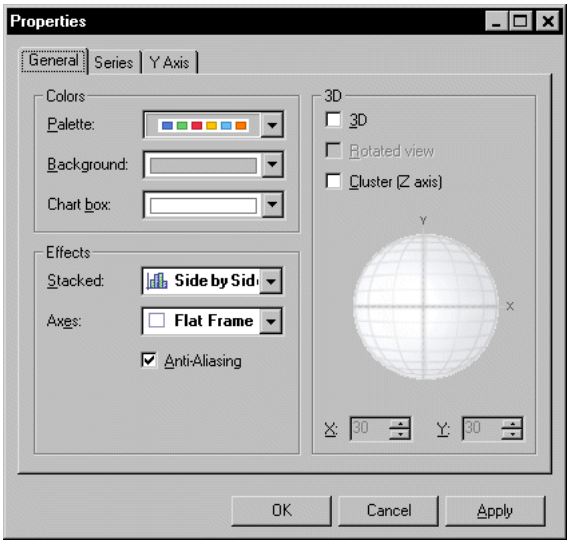
2. Make the formatting changes and click **Apply**.

   Parameters are described in the following section.

3. Close the Properties dialog box.

General Tab

The General tab in the Chart Properties dialog box contains settings that control the colors, effects, and 3D elements.



Following are detailed descriptions of the General chart parameters.

- **Palette**

  Controls the range of colors for all elements in the chart area.

- **Background**

  Controls the background color for the chart. You can also access this option by right-clicking anywhere in the chart and selecting **Color** from the menu that displays.

- **Chart box**

  Controls the color for the inside charting area of the chart, including the back wall of the 3D charts.

- **Stacked**

  Controls how charts are stacked when you have more than one data series in a diagram. You can stack charts side-by-side, on top of each other, or on top of each other and scaled to 100%.

- **Axes**

  Controls the style of the axes. Options are none (no lines), 3D Frame (chart has a grey and white line border so that it looks like 3D), Math (single line for x and y axis), and Flat Frame (chart has a single line border).

- **Anti-Aliasing**

  Controls whether anti-aliasing is used. Anti-aliasing smooths stairsteps to soften jagged edges.

- **3D**

  Controls whether the chart display is 3D or 2D.

- **Rotated view**

  Controls whether the view of the graph is rotated. If you select this check box, move the green dot on the sphere to the location for the axis of rotation. Alternatively, you can enter the x,y coordinates for the rotation in the X and Y text boxes.

- **Cluster (Z axis)**

  Controls whether series are clustered in cases where you have multiple series on one chart. If you select this check box, each data series occupies one row of data and there will be n rows (z-axis clusters) in the chart, where n is the number of series. Area charts are always clustered because there is no way to display different series side by side. However, if you have a bar chart, you can display the bars side-by-side, and not clustered in the z-axis.

- **X**

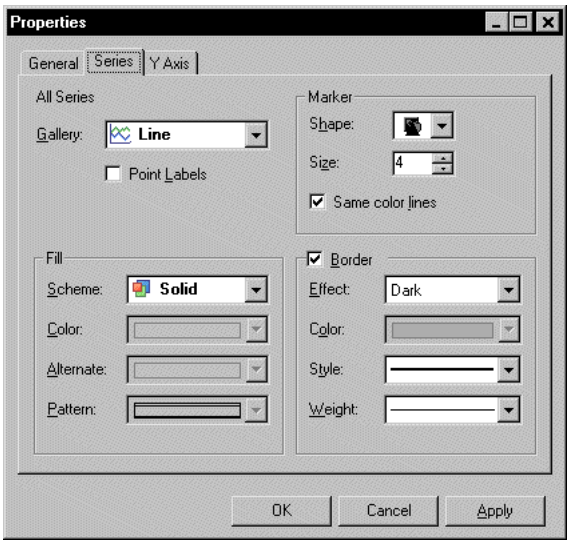
  Controls the 3D rotation angle around the X-axis.

♦ **Y**

Controls the 3D rotation angle around the Y-axis.

Series Tab

The Series tab in the Chart Properties dialog box contains settings that control the style of the series. You can set the properties for each series individually or for all series. To set the properties for a single series, move the mouse cursor over the series before you open the Properties dialog box. To set the properties for all series, move the mouse cursor over a general area of a chart that is not the plot of a series.



Following are detailed descriptions of the Series chart parameters.

♦ **Gallery**

Controls the type of chart used for the data series. In cases where you have multiple series, you can use different types of displays for each series on the same chart.

- **Point Labels**

  Controls whether the values display on the chart for each data point. You can also access this option by right-clicking anywhere in the chart and selecting **Point Labels** from the menu that displays.

- **Scheme**

  Controls the pattern used for the series, such as solid or gradient.

- **Color**

  Controls the color scheme for the series.

- **Alternate**

  Controls the color used for the pattern and gradient schemes.

- **Pattern**

  Controls the pattern used for the pattern schemes.

- **Gap Width**

  Controls the width between bars for bar charts.

- **Square, cylinder, cone**

  Controls whether bar markers for a series display as squares, cylinders, or cones.

- **Border**

  Controls whether borders enclose all data series in the chart.

- **Effect**

  Controls the border effect for a series, such as raised, light, or shadowed.

- **Color**

  Controls the color of the border lines for a series.

- **Style**

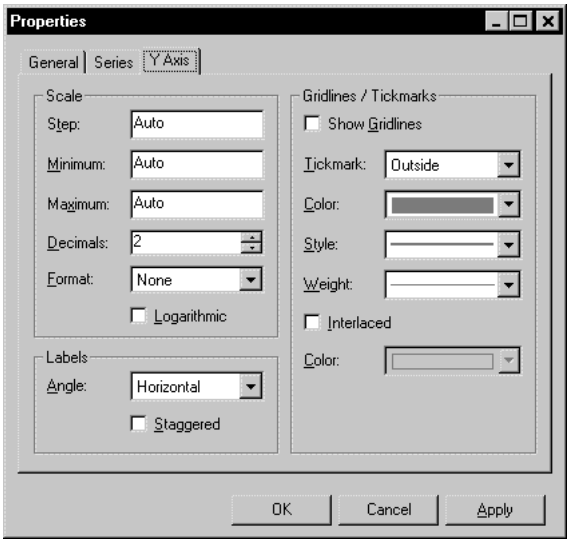
  Controls the style of the border for a series, such as dashed or dotted.

- **Weight**

  Controls the thickness of the border line for a series.

Y Axis Tab

The Y Axis tab in the Chart Properties dialog box contains settings that control the scale, labels, and gridlines on the Y axis. The minimum, maximum, and interval or step values are automatically determined based on the range of data.



Following are detailed descriptions of the Y Axis chart parameters.

- **Step**

  Controls the interval or step of major ticks and gridlines.

- **Minimum**

  Sets the minimum value on the Y axis.

- **Maximum**

  Sets the maximum value on the Y axis.

- **Decimals**

  Controls how many decimal places display for units on the Y axis.

- **Format**

  Controls the format of the values on the Y axis, such as number, currency, scientific, or percentage.

- **Logarithmic**

  Controls whether a logarithmic scale is used on the Y axis.

- **Angle**

  Controls whether the labels are horizontal, vertical, or slanted 45 degrees.

- **Staggered**

  Controls whether labels display with alternate indentations.

- **Show Gridlines**

  Controls whether gridlines display. Gridlines are horizontal lines that cross the chart at every major tick mark.

- **Tickmark**

  Controls the type of tick mark that displays.

- **Color**

  Controls the color of the tick marks and gridlines.

- **Style**

  Controls the line style for tick marks and gridlines, such as solid, dashed, or dotted.

- **Weight**

  Controls the thickness of the tick marks and gridlines.

- **Interlaced**

  Controls whether the chart is interlaced. An interlaced chart has alternating bands of color. Interlacing can help improve readability.

- **Color**

  Controls the second color for the interlaced bands.

Using the Data Editor

The Data Editor displays at the bottom of the chart and contains the values for each of the data points. You can change the values in the Data Editor and the chart updates with the new values.

To change values in the Data Editor:

1. Click in the box containing the value you want to change.

   Delete the old value and then enter the new one.

2. When you move the cursor from the box, the chart refreshes.

To hide the Data Editor:

Right-click anywhere in the chart and select **Data Editor** from the menu that displays.

When the Data Editor is hidden, the Data Editor icon in the right-click menu does not have a border around it.

To display the Data Editor:

Right-click anywhere in the chart and select **Data Editor** from the menu that displays.

When the Data Editor is displayed, the Data Editor icon in the right-click menu has a border around it.

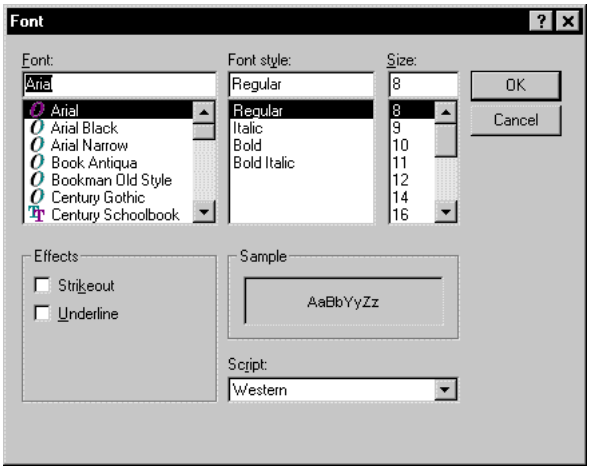
Formatting the Label Text

You can control the size, font, and style of the text used for the labels on the X and Y axis.

To format label text:

1. Right-click the menu anywhere in the chart and select **Font** from the menu that displays.

   The Font dialog box displays.



2. Select the appropriate options and click **OK**.

   The chart refreshes with the new text style.

Formatting the Legend Box

The legend box contains a list of the series plotted on the chart and the color used for each series. You can control the position of the legend box and set the font for the text used in the legend box.

To move the legend box:

Right-click the legend box and select one of the following:

- To position the legend box along the left side of the chart, select **Left** from the menu that displays.
- To position the legend box along the top side of the chart, select **Top** from the menu that displays.
- To position the legend box along the right side of the chart, select **Right** from the menu that displays.
- To position the legend box along the bottom side of the chart, select **Bottom** from the menu that displays.

To set the font for the legend box:

1. Right-click the legend box and select **Font** from the menu that displays.

   The Font dialog box displays.

2. Select the appropriate options and click **OK**.

   The legend refreshes with the new text style.

Copying Charts

The Charts Copy command copies the displayed chart to Windows clipboard. Once the chart is copied to the clipboard, you can use the Paste command to insert the chart into another application.

To copy a chart:

1. Display the chart.

   For details, see “Adding Charts” on page 184.

2. Display the Chart toolbar by right-clicking anywhere in the chart and selecting **Toolbar**.

3. On the Chart toolbar, click the **Copy to Clipboard** button.

4. Select one of the following from the drop-down menu that displays:
   - To copy the chart as a bitmap, select **Bitmap**.
   - To copy the chart as a metafile, select **Metafile**. Metafiles are a Windows image format.
   - To copy the chart as a text, select **As Text (data only)**. Only the values of the data points for each series are copied.

   The chart is copied to the clipboard and can now be pasted.

Pasting Charts

To paste a chart:

1. Open the application in which you want to paste the chart.

2. Switch to Micro Saint Sharp and copy the chart.

   For details, see “Copying Charts” in the previous procedure.

3. Use the Paste command in the secondary application to paste the chart.

Printing Charts

To print a chart:

1. Display the chart.

   For details, see “Adding Charts” on page 184.

2. From the **File** menu, select **Print Chart**.

   The chart is printed.

Saving Charts As an Image File

To save a chart as an image file:

1. Display the chart.

   For details, see “Adding Charts” on page 184.

2. From the **File** menu, select **Save Chart as Image**.

   The Save As dialog box displays.

3. Select the folder where you want to save the chart and enter a name.

4. Click **Save**.

   The chart is saved to the specified location.

10

Animator

Using the Animator feature, you can build two-dimensional image animation into a model. As you run the model, images appear, move, and disappear against a customized background representing the processes that occur in the model.

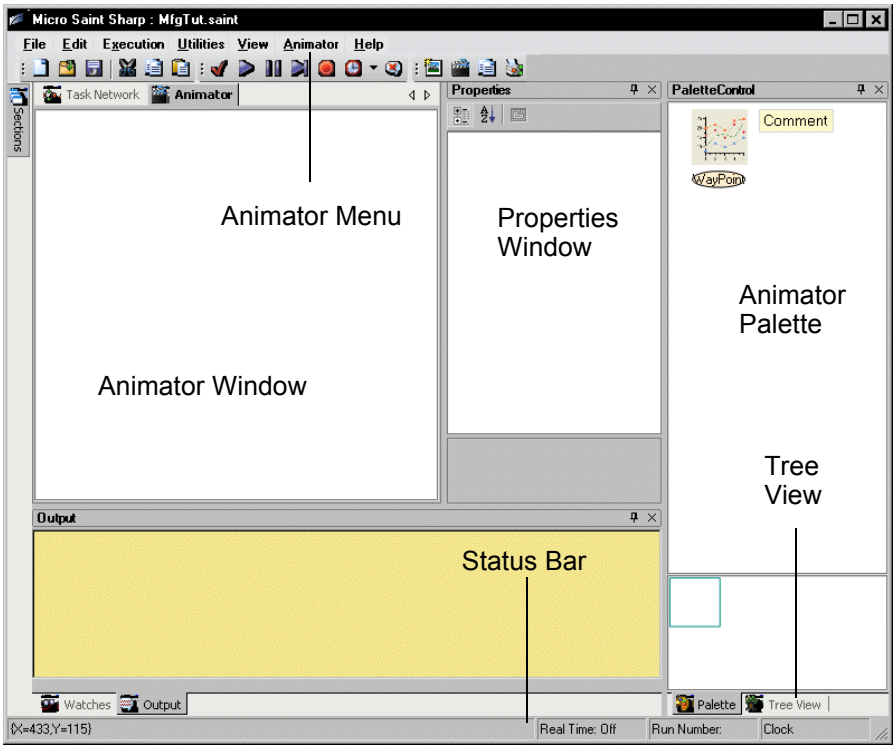
The Animator display can contain a background, images that move across the background, comments, way points, and charts. Comments allow you to add text such as a heading or label to the display, and additionally add watches for any variables. Way points allow you to specify multiple paths from a given point. And finally, charts allow you to dynamically monitor data as it is recorded during the simulation. You can use Animator functions to manipulate the images, image text, background, way points, and comments.

You can copy and paste the Animator display into another application and print the display.

Note:
Animator is not automatically included with Micro Saint Sharp; you must purchase the Animator module separately.

Animator Window Overview

The Animator window consists of the container window with the following components: main Animator window, Animator palette, tree view, Animator menu, Properties window, and status bar.



Animator Window

The Animator window is the main window where you construct and view the Animator diagram. You create the diagram by adding images, charts, comments, and way points.

Animator Palette

The Animator palette displays the tools that you use to construct the animation. Tools for adding images, way points, graphs, and comments are included.

- Images. Use images to represent the background of the display and the movement of entities through the model.
- Way Points. Use way points to indicate where paths may split and are are multidirectional.
- Charts. Use charts to display data values during the animation. Available charts are the ones you created for the model.
- Comments. Use comments to add clarification to the Animator display and to display values of variables on the Animator display as the model runs.

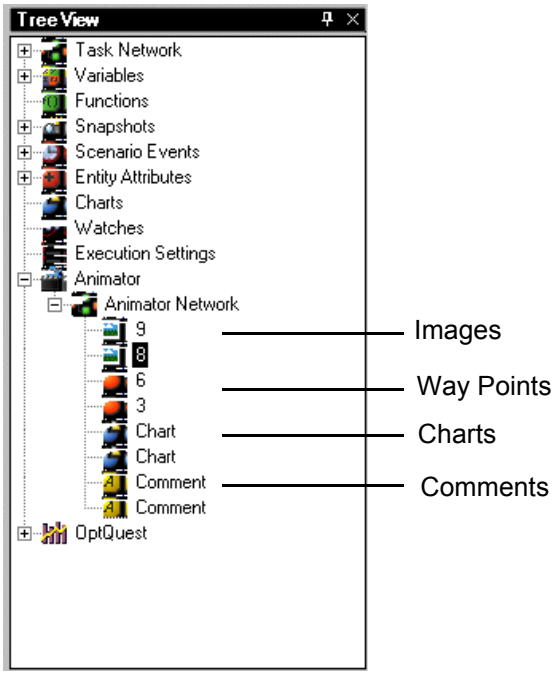
The relative position of the active display in the Animator window diagram displays at the bottom of the Animator palette.

Tree View

The list of model components in the tree view contains the Animator object. Expand the Animator object to view the images, comments, charts, and way points in the Animator display.

To display a description window for an Animator item, double-click the item in the tree view.

Animator elements are represented by the following icons.



Animator Properties Window

The Properties window displays the properties of the item selected in the Animator display, such as an image, comment, chart, or way point. The Properties window functions like any folder list. To expand an item, click the plus sign adjacent to the item. To collapse an item, click the minus sign adjacent to the item.

A detailed description of any selected item automatically displays at the bottom of the Properties window.

To sort the items in the property sheet, do one of the following:

- To sort the properties alphabetically, click the **Alphabetic** button .
- To sort the properties by category, click the **Categorized** button .

Animator Menu

The Animator menu contains the following commands:

- Add Image to Animator Palette. Use this command to add images to the Animator Palette. You must add images to the palette before they can be added to the Animator display.
- Animator Edit Mode. Use this command to modify the Animator display.
- Copy Animator Display. Use this command to copy the Animator display.
- Print Animator Display. Use this command to print the Animator display.

Animator Status Bar

The status bar displays at the bottom of the Micro Saint Sharp window. Any execution-related messages display on the left side of the status bar The clock values during execution display on the right side of the status bar.

Note:
You can click anywhere on the Animator window and the coordinates display in the status bar.

Displaying the Animator Window

If the Animator window does not currently display, use the following command to open it.

To display the Animator window:

1. From the **View** menu, select **Windows**.
2. Select **Animator**.

   The Animator window displays.

Enabling Edit Mode

The Animator uses two modes: Edit mode and Run mode. Animator automatically switches to Run mode when a model is running. In this mode, you cannot change the display. You must switch to Edit mode to modify the Animator display. Once a model has completed running, the Animator is left in Run mode until you switch to Edit mode. This allows you to copy or print the Animator end states.

To enable Edit mode:

From the **Animator** menu, select **Animator Edit Mode**.

The Animator display changes to Edit mode and you can modify the diagram.

Adding Objects to the Animator Window

To add an object to the Animator window using the Animator palette:

1. From the Animator palette, select the item to add.

   When you select an item, a green box displays around the item.

2. Click the left mouse button, and while pressing the button, drag the item to the Animator display.

3. Release the left mouse button.

   The object is added.

To add an object directly in the Animator window:

1. Right-click the mouse in the Animator window.

2. Select **Add Comment**, **Add Graph**, or **Add Way Point** from the menu that displays.

3. Resize the object to the appropriate size.

   You can resize any of the items except for way points.

Positioning Objects Vertically

You can position overlapping images in a vertical sense. The Layer parameter on the Properties window determines the "vertical" position for an image in a layer.

For objects that overlap, you can use the following commands:

- To move an object to the bottom of a group of objects, select the item, right-click the mouse, and select **Move to Bottom** from the menu that displays.

- To move an object to the top of a group of images, select the item, right-click the mouse, and select **Move to Top** from the menu that displays.

- To move an object below another object, select the item, right-click the mouse, and select **Move Down** from the menu that displays.

- To move an object above another object, select the item, right-click the mouse, and select **Move Up** from the menu that displays.

Working With Images

You use images to represent the movement of entities in the simulation. You can use a wide variety of image types with the Animator display. The following image types are supported: `.BMP`, `.GIF`, `.JPG`, `.PNG`, `.TIFF`, and `.ICO`.

Adding Images to the Animator Palette

Before you can add images to the Animator display, you must first add them to the Animator palette.

To add images to the Animator palette:

1. From the **Animator** menu, select **Add Image to Animator Palette**. You can alternatively right-click in the Animator Palette and select **Add Image** from the menu that displays.

   The Add Image dialog box displays.

2. Navigate to the location containing the image and select the image.

3. Click **Open**.

   The image is loaded into the Animator palette. Images are saved with the model.

Deleting Images from the Animator Palette

You can delete an image from the Animator palette when you no longer need it. If you delete an image from the palette that is in the Animator display, the image is also removed from the Animator display.

To delete an image from the Animator palette:

1. In the Animator palette, select the image.

   A green box displays around an item when it is selected.

2. Press the Del key on the keyboard. You can alternatively right-click the image in the Animator display and select **Delete Object** from the menu that displays.

Adding Images to the Animator Display

To add an image to the Animator display:

1. Click the Animator palette.

2. Select the image you want to add.

   When you select an item, a green box displays around the item.

3. Click the left mouse button, and while pressing the button, drag the item to the task network diagram.

4. Release the left mouse button.

   The image is added and a Properties window displays. The Properties window for the image displays the following information.

   - Angle. Displays the rotation angle for the image.
   - Filter Color. Displays the color filter to apply to the image.
   - Height. Displays the height of the image in pixels.
   - Width. Displays the width of the image in pixels.
   - Center X. Displays the x coordinate position for the center of the image. The upper left corner of the Animator display is considered to be (0,0)
   - Center Y. Displays the y coordinate position for the center of the image.
   - Layer. Displays the z coordinate position for the image.
   - Name. Contains the name of the image (optional).
   - Notes. Contains any notes regarding the image.

- Unique ID. Contains a unique ID for the image as assigned by Micro Saint Sharp. This is the ID you would use in Animator functions. You can use the same image in multiple locations, but each image will have a different ID.
- Type ID. Contains the ID for the image as assigned in the Animator palette. Type ID is an ID shared by all copies of a particular palette image ID.

5. Resize the image to the size that you want.

   To resize an image, click the image. Small boxes or handles display at the corners and along the sides. Select one of the handles, and drag the box to resize the image while pressing the left mouse button.

Identifying Images in the Animator Display

Two IDs are important for identifying images in the Animator display. These IDs display in the Properties window for each image.

- Unique ID is the unique ID for the image as assigned by Micro Saint Sharp.You can use the same image in multiple locations, but each image will have a different ID.

- Type ID is the ID for the image as assigned in the Animator palette. Type ID is an ID shared by all copies of a particular palette image ID.

Setting the Background of the Animator Display

You can add an image to the Animator display that serves as a background. When an image is loaded as a background, you can place other images on top of it. You can only have one background active at a time. If you load a second background, the first one is automatically deleted from the Animator window (not from the Animator palette).

To set the background of the Animator display:

1. Right-click in the Animator display and select **Set Background** from the menu that displays.

   The Open dialog box displays.

2. Navigate to the location containing the background image and select the image.

3. Click **Open**.

   The image is loaded into the Animator display and into the Animator palette. Resize the background if you would like it to cover more area.

Clearing the Background of the Animator Display

To clear the background of the Animator display:

Right-click in the Animator display and select **Clear Background** from the menu that displays.

The background is removed.

Charts in the Animator Display

You can add charts to the Animator display. Charts are created in the same way as for the network diagram; see "Adding Charts" on page 184 for details. The main difference with Animator charts is that you can place and size the graphs on the Animator display. In this way, you can see images moving on the Animator display and additionally view the data and charts in the same window.

The Chart Description dialog box contains an additional tab for charts in the Animator display. The Animator tab contains the Unique ID, Center X, Center Y, and layer values for the chart.

Adding Charts to the Animator Display

To add a chart to the Animator display:

1. Drag the Chart tool from the Animator palette onto the Animator display to the location where you want it. You can alternatively right-click the mouse and select **Add Graph** from the menu that displays.

2. Resize the chart to the appropriate size.

   To resize a chart, click the chart. Small boxes or handles display at the corners and along the sides. Select one of the handles, and drag the box to resize the chart while pressing the left mouse button.

3. On the Animator display, double-click the chart.

   The Chart Properties dialog box displays.

4. Set the Chart properties.

   For details on chart parameters, see “Adding Charts” on page 184.

Deleting Charts from the Animator Display

To delete a chart from the Animator display:

1. In the tree view, expand the Animator object and select the chart.

2. Select the object and press the Del key on the keyboard. You can alternatively right-click the chart in the Animator display and select **Delete Object** from the menu that displays.

Way Points

You can use way points to specify more than one path for an image to follow.

Adding Way Points

To add a way point:

1. Click the Animator palette.
2. Select the way point tool.

   When you select a tool, a green box displays around the item.
3. Click the left mouse button, and while pressing the button, drag the way point to the Animator display and place it where you want it.
4. Release the left mouse button.

   You can alternatively right-click the mouse in the Animator display and select **Add Way Point** from the menu that displays.

   The way point is added and a Properties window for the way point displays. Micro Saint Sharp automatically numbers way points consecutively. The following information is included in the Properties window:

   - Center X. Displays the x coordinate position for the center of the way point. The upper left corner of the Animator display is considered to be (0,0)
   - Center Y. Displays the y coordinate position for the center of the way point.
   - Layer. Displays the z coordinate position for the way point.
   - Notes. Displays any notes regarding the way point.
   - Unique ID. The unique ID for the way point.

- Background Color. Displays the background color for the way point.
- Border Color. Displays the border color for the way point.
- Font Size. Displays the font size for the way point.
- Font Type. Displays the font face for the way point.
- Text Color. Displays the text color for the way point.

Deleting Way Points

To delete a way point:

1. Select the way point.

2. Press the Del key on the keyboard. You can alternatively right-click the way point in the Animator display and select **Delete Object** from the menu that displays.

Comments

You can use comments to add clarification to the Animator display and to display values of variables on the Animator display as the model runs. For example, you could place the name, description, and last revision date of the animation as a title on the Animator display.

You can position comments in the same manner as any other items on the Animator display. A list of comments associated with the Animator displays in the tree view below the Animator node. You can display comment descriptions, and add, edit, and delete comments.

Displaying Comment Descriptions

To display a comment description:

Double-click the comment on the Animator display.

The Comment Description dialog box displays in the main Document window and the associated Properties window for the comment displays.

Adding Comments

To add a comment:

1. From the Animator palette, drag the Comment icon to the correct place on the network diagram.
2. Double-click the comment box.

   The Comment dialog box displays.
3. Enter the comment parameters.

   For details on Comment tabs and parameters, see “Comment Parameters” on page 153.

Editing Comments

To edit a comment:

1. Double-click the comment in the Animator display.

   The Comment Description dialog box displays in the Animator window and the Properties window for the comment also displays.
2. Make your changes in the Comment dialog box or Properties window.

Deleting Comments

To delete a comment:

1. On the Animator display, select the Comment box.

2. Press the Del key. You can alternatively right-click the comment in the Animator display and select **Delete Object** from the menu that displays.

   The comment is removed.

Using Animator Functions

Micro Saint Sharp provides a wide variety of Animator functions that you can use to control the animation. You enter these functions in the Task Description dialog boxes in the task network. A task description contains beginning, ending, and launch effects. The beginning effect is executed when the task begins execution and the ending effect is executed when the task finishes execution. The launch effect is often used to execute expressions that cause Animator events.

The launch effect is similar to a beginning effect except that it is evaluated after the task execution time, or the `Entity.Duration`, is calculated. The Launch Effect field usually contains the function calls describing the image movements. The launch effect allows the Animator and task network to remain in synchronization by having the image movements last as long as the task does.

Micro Saint Sharp provides the following types of Animator functions: Image, Image Text, Background, Way Point, and Comments. For details on Animator functions and function parameters, see “Animator Functions” on page 281.

Developing an Animator Display

The following procedure summarizes how to develop an Animator display.

To develop an Animator display:

1. Open the Animator window.
2. Add the images that you want to use to the Animator palette.

   For details, see "Adding Images to the Animator Palette" on page 218.
3. From the Animator palette, select the item you want to add.

   When you select an item, a green box displays around the item.
4. Click the left mouse button, and while pressing the button, drag the item to the Animator display.
5. Release the left mouse button.

   The object is added to the Animator window. Do any of the following:

   - Reposition the item to the correct location.
   - Resize any images to the size that you want. For details, see "Adding Images to the Animator Display" on page 219.
   - Add any charts you want to include and enter the chart parameters. For details, see "Adding Charts to the Animator Display" on page 222.
6. Repeat steps 3 to 5 for all the objects you want to add to the Animator display.

7. Add any Animator functions to the task network task descriptions to control the images, image text, background, way points, and comments.

8. Run the simulation.

Executing the Animator Simulation

To execute the Animator simulation, you run the model simulation. When running Animator, you must run the simulation in Realtime mode for the animation to synchronize correctly with the simulation (see page 161).

To execute the Animator simulation:

1. Open the Animator display.

2. From the **Execution** menu, select **Begin Simulation** or press Ctrl-G.

   The model executes. For details on setting simulation speed and other execution parameters, see “Running the Model” on page 157.

Copying the Animator Display

You can copy the Animator display and paste it into another application. The Animator display is copied as an image.

To copy the Animator display:

1. Open the Animator window.

2. From the **Animator** menu, select **Copy Animator Display**.

   The Animator display is copied and you can paste it into another application.

Printing the Animator Display

To print the Animator display:

1. Open the Animator window.
2. From the **Animator** menu, select **Print Animator Display**.

   The Print Preview window displays. For details on the options in the window, see “Printing the Task Network Diagram” on page 86.
3. Click the **Print** button in the Print Preview window.

■ Printing the Animator Display

11

Expressions

A Micro Saint Sharp expression is a calculation, formula, function, or statement that supplies a value or performs an operation. You can use expressions to supply numeric values such as mean times or true/false values such as release conditions. You can also make changes in the state of a model, such as beginning effects.

Note**:**
Expressions in Micro Saint Sharp use Microsoft C# syntax. For additional details on syntax, consult Microsoft C# documentation.

Elements in Expressions

Each expression in Micro Saint Sharp must end with a semi-colon. Expressions can include the following elements: constants, variables, functions, comments, commas, return statements, arithmetic operators, assignment operators, adjustment operators, logical operators, and if-else, while, for loops, do, switch, and foreach statements.

Constants

A constant is a quantity with a fixed, unvarying value—usually a number. Often in a model you supply expressions that are constants for numeric task descriptors such as the mean time and standard deviation, or for logical task descriptors that are always true (like the default release condition) or always false.

Constants are also often used in combination with variables and mathematical or logical operators. In the release condition, the value of a variable is compared with a constant. In the mean time and standard deviation, constant values are multiplied by variable values as adjustments.

Expressions that are constants	Expressions that contain constants
`{Release Condition}` `return 1;`	`{Release Condition}` `operators > 0;`
`{Mean Time}` `return 9;`	`{Mean Time}` `15 *(1 + FatigueFactor);`
`{Standard Deviation}` `return 1.5;`	`{Standard Deviation}` `2 * (1 + Temperature/2);`

Variables

A variable is a string of characters that is assigned a numeric value. Variable names must begin with a letter or underscore and can be followed by zero or more letters, numbers, and underscore characters. Valid examples include `abc`, `_abc`, `_aA12`.

Variable names are case-sensitive, so name and NAME are considered to be different variables.

The Microsoft C# language reserves 76 words for its own use. Do not use any of these keywords as variable names. For a list of reserved names, see page 116.

You can change the value assigned to a variable at any time using the assignment operator (=) or one of the adjustment operators (+=, -=, *=, /=). In the following example, the variable `Operators` (representing the number of operators available to perform the task) decreases by one when the task starts and increases by one when it finishes. You can use a variable in any place that you would use a constant in an expression.

```
//Beginning Effect
Operators -= 1;  or  Operators = Operators - 1;

//Ending Effect
Operators += 1;  or  Operators = Operators +1;
```

Variable Scoping

Global variables must be defined in the variable list in Micro Saint Sharp. Local variables do not need to be defined in the variable list but are defined inside of a code section (for example, Beginning Effect or Queue Priority Effect). Local variables can be used to hold temporary values. An example might be when you are using a for loop. Rather than defining a global variable that iterates the loop, you could use a local variable as the loop iterator.

For example,

```
for(int i=0; i<10; i++)
{
  Model.PrintOutput("i: " + i);
}
```

The variable `i` is defined as an integer at the beginning of the loop and is accessible as long as the loop is running.

Functions

You can call a function in an expression to perform the procedure associated with the function or to return a value. All functions must end with a set of parentheses; this includes entity and user-defined functions.

Function names must begin with a letter or underscore and can be followed by zero or more letters, numbers, and underscore characters. Function names are case-sensitive so name and NAME are considered to be different functions.

You can use functions that return numeric values in any place that you would use a numeric constant. Functions that perform procedures are generally used as single expressions, as the “then” part of an if-then statement, or as the “do” part of a while loop.

```
//Beginning Effect
 if (Clock == 1440)
  {
       Model.Halt();
  {

//Mean
 Distributions.Poisson(15);
```

Comments

When a model includes long or complex expressions, you may want to add explanatory comments. Micro Saint Sharp allows two types of comments:

- Single line comments. In single line comments, the text between // and a carriage return is disregarded.

```
//Function AVERAGE calculates the average.
```

- Multi-line comments. In multi-line comments, the text between /* and */ is disregarded.

```
/*SUMEXPOSE counts and sums the nonzero
elements in the array exposure putting the
count in j and the sum in sumexpose*/
```

Commas

Commas are not allowed in Micro Saint Sharp expressions except for in arrays. You can use commas in arrays if you have a two-or-more dimensional array, for example `int[,] a = new int[12,14];`

Return Statements

A return statement returns a value so that it can be used in another calculation or evaluation. Return statements must be included in several instances in Micro Saint Sharp:

- When an expression in a model contains a variable or value used in other calculations. In this case, the last expression must begin with return.

  Example: The following expression returns the value of `b`.

  ```
  return b;
  ```

- When logic statements are evaluated in the Release Condition, Decision Code, and Queue Priority fields. In this case, a value of true or false must be returned. If a decision is of type probabilistic, a decimal value must be returned.

  Example: The following expression could be used in the Release Condition field.

  ```
  if (x>2)
    {
      return true;
    }
  else
    {
      return false;
    }
  ```

  The task would only execute when `x` was greater than 2.

- When expressions are evaluated in the Mean Time, Standard Deviation, or the Third Data Shaping fields. In this case, a number must be returned.

  Example: The following expression could be used in the Mean Time field.

```
if (Entity.ProductType == 1)
  {
   return 5;
  }
else
  {
   return 7;
  }
```

When user-defined functions are evaluated (excluding void types) the value of the function is automatically returned. A return statement is not required.

Arithmetic Operators

You can use the operators in the following table to perform arithmetic operations.

Operator	Purpose	Example
( )	Grouped operations; items in parentheses are executed first	(2*3) +1 = 7 but 2 * (3+1) = 8
*	Multiplication	2*3 = 6
/	Division	6/2 = 3

■ Arithmetic Operators

Operator	Purpose	Example
%	Remainder division (also called mod, modular modulo, or modulus); only integers can be used with this operation.	7 % 2 = remainder of 7/2 = 1 12%2 = remainder of 12/2 = 0
-	Subtraction	22 - 11 = 11
+	Addition	11 + 22 = 33
++	Increment add 1 to the value of an integer	`x++`
--	Decrement subtract 1 from the value of an integer	`x--`

The order of precedence for these operations is from the top to the bottom, that is, exponentiation is performed before multiplication, and so on.

To change the order in which operations are performed, use parentheses. Operations within parentheses are performed first. Inside parentheses, the usual order of operations is maintained.

Note:
If you are unsure how Micro Saint Sharp evaluates the mathematical operators, use parentheses to clarify the order.

Assignment Operator

The assignment operator (=) assigns the value on the right of the operator to the variable on the left of the operator. For example, the expression `a = 3;` assigns the value three to the variable `a`. It also returns the value three, which is nonzero, so it would be considered a true statement if you used it as a logical condition. However, you would generally not want to use the assignment operator in a logical condition statement.

Be careful not to confuse the assignment operator with the logical equality operator (==). To test for equality, use the logical equality operator (==). This returns a value of one if the value on the right of the operator equals the value on the left, and returns a value of zero if it does not.

`a = 3;` means set a equal to 3

`a == 3;` means does the value of a equal 3?

Adjustment Operators

The adjustment operators adjust the value of a variable based on its current value and some other value. These operators provide a shorthand for expressions that require repeating the variable name and using the assignment operator, as shown in the following table. You can use either of these forms when you need to adjust the value of a variable.

Long form	Short form	Example
variable = variable + number;	variable += number;	If x = 2, then `x += 3` would set x to 5
variable = variable - number;	variable -= number;	If x = 2 then `x -=3` would set x to -1
variable = variable * number;	variable *= number;	If x = 2, then `x *2 =` sets the value of x to 4

Long form	Short form	Example
variable = variable / number;	variable /= number;	If x = 2, then `x /= 2` sets the value of x to 1
variable = variable % number;	variable %= number;	If x=2, then `x% = 2` sets the value of x to 0.

Logical Operators

Use logical operators to compare two numeric values or logical statements. If the comparison is true, the result is equal to one. If the comparison is equal to false, the result is equal to zero. Similarly, any nonzero value is considered true and a zero value is considered false.

✓ Note:
Be careful not to confuse the logical equality operator (==) with the assignment operator (=). The assignment operator sets the variable to its left equal to the value to its right and returns this value.

Symbol	Comparison	Example
==	Equal to	`a == 3` If the value of a is 3, the expression is true and returns a value of one. If the value of a is anything other than 3, the expression is false and returns a value of zero.
!=	Not equal to	`a!=5` If a is not equal to 5, the expression is true and returns a value of one. If a is equal to 5, the expression is false and returns a value of zero.

Symbol	Comparison	Example
<	Less than	`a<5` If a is less than 5, the expression is true and returns a value of one. If a is greater than 5, the expression is false and returns a value of zero.
<=	Less than or equal to	`a<=5` If a is less than or equal to 5, the expression is true and returns a value of one. If a is greater than 5, the expression is false and returns a value of zero.
>	Greater than	`a>5` If a is greater than 5, the expression is true and returns a value of one. If a is greater than 5, the expression is false and returns a value of zero.
>=	Greater than or equal to	`a>=5` If a is greater than or equal to 5, the expression is true and returns a value of one. If a is greater than 5, the expression is false and returns a value of zero.
&&	Logical AND	`a==3 && b==4` If the value of a is 3 and the value of b is 4, the expression is true and returns a value of one. If the value of a is anything other than 3, the expression is false and returns a value of zero.

Symbol	Comparison	Example
`||`	Logical OR	`a==3 || b==4` If the value of a is 3 or the value of b is 4, the expression is true and returns a value of one. If the value of `a` is anything other than 3, the expression is false and returns a value of zero.
`!`	Not	`!(a==3)` If a is not equal to 3, the expression is true and returns a value of one. If a is equal to 3, the expression is false and returns a value of zero.

If-Else Statements

Sometimes in a model, you may want to perform certain expressions only under specific conditions. For these cases, Micro Saint Sharp uses the if-else statement. This statement can be used in several different ways.

- If statements. These take the following format:

  ```
  if (expression) {code to execute};
  ```

  Example:

  ```
  if (x==3)
  {
    y = 5;
  }
  ```

- If-else statements. These take the following format:

```
if (expression to evaluate) {code to execute}
else {code to execute}
```

Example:

```
if(x==3)
{
  y=5;
}
else
{
  z = 3;
}
```

When you have multiple expressions, you can include the lines in braces, such as in the following:

```
if (x == 5)
{
   b = 1;
   c = 1;
}
```

```
if(x == 3)
{
 y =5;
}
```

```
else
{
 z = 3;
}
```

You can additionally nest if statements, such as in the following example:

```
if(x == 3)
{
 if (y==4)
 {
  z = 3;
 }
}
```

- Cascading If Statements. You can nest if statements inside other if statements. This way you can chain together a sequence of boolean expressions that will be tested one after the other until one evaluates to true.

  For example,

```
if (day==0 | | day==7)
  dayString = "Weekend";
else if (day>1 && day <=6)
  dayString = "Weekday";
else
  dayString ="unknown";
```

While Statements

The while statement executes a block of statements repeatedly until a specified expression evaluates to false. While statements use the following format:

```
while (expression to evaluate) {expressions to
execute}
```

For example, in the following statements, as long as the value of x is less than 10, the value 1 is added to x.

```
while (x<10)
{
  x++;
}
```

When a while statement is executed, Micro Saint Sharp first checks whether the expression to evaluate is true; if so, it executes the following expression. Then, it checks the initial expression again; if it is still true, Micro Saint Sharp executes following expression again. This process continues until the initial expression becomes false. When Micro Saint Sharp determines the initial expression is false, it does not execute the following expression again. The initial expression should always be one that eventually becomes false; otherwise you have an infinite loop that does not stop processing.

For Loops

The for loop executes a block of statements repeatedly until a specified expression evaluates to false. A for loop is similar to a while statement, but iterates a set number of times. The for loop is useful when you know how many times a loop needs to be executed. For loops use the following format:

```
for (initializer; condition; iterator value)

{expressions to execute}
```

The initializer sets the initial value of the iterator, the condition controls whether the execution occurs, and the iterator value either increments or decrements the initializer value. The expressions execute as long as `condition` is true.

For example, in the following statements, as long as the value of `j` is less than 10, the value 1 is added to `j` and the number of workers is decreased by one.

```
for (int j=0; j<10; j++)
{
  workers --;
}
```

When a for loop is executed, Micro Saint Sharp first checks whether the condition is true; if so, it executes the following expression. Then, it checks the condition again; if it is still true, Micro Saint Sharp executes following expression again. This process continues until the condition becomes false. When Micro Saint Sharp determines the condition is false, it does not execute the following expression again. The condition should always be one that eventually becomes false; otherwise you have an infinite loop that does not stop processing.

Do Statements

The while and for statements both test the boolean expression at the top of the loop. Therefore, if the expression is false on the first test, the body of the loop will not execute. There may be instances where you want the body to run once. For these cases, use the do statement. The syntax of a do statement is the following:

```
do
  {expression;}
   while (boolean expression);
```

Switch Statements

Switch statements can be used in place of cascading if statements (see page 244). They enable your code to be more efficient and readable. The syntax of a switch statement is the following:

```
switch (control Expression)
{
  case constant Expression:
    statement;
    break;
  case constant Expression2:
    statement;
    break;

...

default:
    statement;
    break;
}
```

■ Switch Statements

Switch statements must adhere to the following rules:

- You can only use switch on int and string data types.
- The case labels must be constant expressions such as 3 or “3”.
- The case labels must each have a unique expression; no two can be the same.
- A `break;` statement is required for each case.
- A default case is allowed.

For example:

```
string fruit = “apple”;
switch (fruit)
{
    case “banana”:
   {
     Model.PrintOutput (“You ate a banana”);
     break;
   }
    case “apple”:
   {
     Model.PrintOutput (“You ate an apple”);
     break;
   }
    default:
   {
     Model.PrintOutput (“didn’t eat fruit”);
     break;
   }
}
```

Foreach Statements

Sometimes you may want to loop through all the items in an array or some other collection of objects. In this case, you can use a foreach statement. Foreach statements take the following format:

```
foreach (type identifier in expression)
}
   statement;
}
```

For example:

```
foreach (int current in MyArray)
{
   Model.PrintOutput ("current:" + current);
}
```

■ Foreach Statements

12

Built-in Functions

Micro Saint Sharp includes a large number of built-in functions that you can use to calculate values. When Micro Saint Sharp encounters the function, it executes the function and returns a value. The built-in functions in Micro Saint Sharp include the following categories:

- Modeling functions, which perform actions such as starting or stopping tasks, and pausing or stopping model executions.
- Mathematical functions, which perform mathematical operations, such as minimum, maximum, and trigonometric functions.
- Distribution functions, which create probability distributions.
- Animator functions, which create and move Animator images and control the attributes of the Animator display (Animator is an optional add-on component).

Function Syntax and Usage

A Micro Saint Sharp built-in function consists of a word identifying the function type (Model, Math, Distribution, or Animator), followed by a period, and then followed by the function name.

If the function requires values supplied to it, include the values in parentheses following the function name; otherwise, type the parentheses but leave them empty. The values you supply in the parentheses can be numbers, variables, or expressions.

Micro Saint Sharp provides assistance when you type a function name in any expression text box:

- After you type the name of the function type and the following period, Micro Saint Sharp automatically displays a list of available functions and properties for your selection. Functions are indicated by a purple, diamond-shape icon and properties are indicated by a grey, rectangular-shape icon. Properties are a type of Model function that retrieve execution properties such as `RunNumber` and `RandomSeed`. Double-click a function or property to select it.

Note:
Functions require parentheses and may additionally require other parameters. Properties do not require parentheses and do not take parameters.

- After you type the name of a function and the left parentheses, Micro Saint Sharp automatically displays a tip box with a description of the function, the variable type for the function, and the required function parameters. Click the mouse to close the description.

You can use a function anywhere in a model where you need to supply a value or an expression. A few examples are shown in the following list:

```
//Mean Time
 Model.Poisson(15);

//Beginning Effect
 if (Clock == 1440)
  {
    Model.Halt();
  {

//Launch Effect
 Animator.MoveImage(7,10,20,Entity.Duration);

//Ending Effect
 MyFunction( );
```

Model Functions

Model functions control model execution, and can perform actions such as halt, pause, abort, or suspend. All Model functions start with the word Model, followed by a period, and then followed by the function name, such as `Model.Halt( )`.

✓ Note:
Function names are case-sensitive.

For some of the model functions, you supply an attribute for the entity. The attribute can be a default attribute or a user-defined attribute. Default attributes consist of the following:

- `Entity.Duration` records the task execution time for the entity.
- `Entity.Tag` records the tag ID number of the entity. The `Entity.Tag` value is an integer that records the identity of each entity when multiple entities travel through a network.

- `Entity.Group` is similar to the `Entity.Tag` attribute, but is used to control groups of tasks.

  For example, the aircraft in a simulation are given assigned `tag` values, but have scanning tasks that are assigned to a tank commander, gunner, and the loader. When an aircraft is detected, you want to stop the scanning tasks and react to the hostile aircraft. Using the attribute `Entity.Group`, you can label the tank commander's scanning task as `Entity.Group = 1`, the gunner as `Entity.Group = 2`, and the loader as `Entity.Group = 3`. Then you can control whether the group of tasks are active through group attributes. As with the `Entity.Tag` attribute, the `Entity.Group` value is carried through the simulation.

- `Entity.ID` records the current task number the entity is in. The task number is assigned to the task by Micro Saint Sharp, but you can change the number in the task ItemID box.

- `Entity.Time` contains the closest scheduled event time for the entity.

- `Entity.Type` contains the event type for which the entity is scheduled. These can be one-time events or they can repeat at regular intervals.

Abort

Syntax: Abort("key", value)

Description: Aborts all entities matched by the key-value pair. The key parameter can be a default entity attribute or a user-defined attribute.

Example: `Model.Abort("Tag",3)` aborts any entities with a `tag` value of 3.

`Model.Abort("ID", 2)` aborts any entities in task ID 2.

✓ Note:
The ending effect of a task that is aborted is not executed.

Abort (array)

Syntax: Abort("*key array", value array*)

Description: Aborts all entities matched by the key-value pair. The key parameter can be a default entity attribute or a user-defined attribute.

Example: If key[0] = "Group," key[1] = "Tag," val[0] = 3, and val[1] = 12, then `Model.Abort(key, val)` aborts any entities with a group value of 3 and a tag value of 12.

If key[0] = "Group," key[1] = "ID," val[0] = 3, and val[1] =12 then `Model.Abort("Tag",3)` aborts any entities with a tag value of 3. `Model.Abort("ID", 2)` aborts any entities in task ID 2.

✓ Note:
The ending effect of a task that is aborted is not executed.

Beep

Syntax: Beep(*number*)

Description: Causes a beeping noise during model execution. You indicate the number of beeps.

Example: `Model.Beep(3)` causes three beeps to sound.

Create Entity

Syntax: CreateEntity( )

Description: Creates an entity with attributes that are all set to zero.

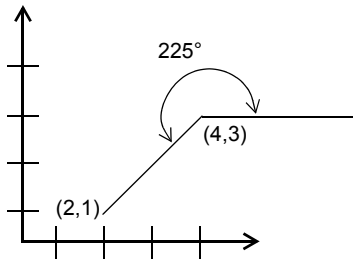
Example: `Entity TemporaryEntity=CreateEntity( )` points the entity variable `TemporaryEntity` to a new entity with attributes that are all equal to zero.

Direction

Syntax: Direction(*x1*, *y1*, *x2*, *y2*)

Description: Returns the direction (in degrees) between two points (x1,y1) and (x2,y2).

Example: `Model.Direction (4,3,2,1)` returns 225.

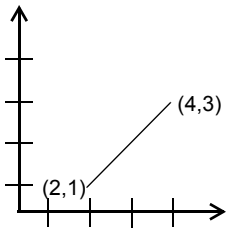


Distance

Syntax: Distance (*x1*, *y1*, *x2*, *y2*)

Description: Returns the distance between two points (x1,y1) and (x2,y2)

Example: `Model.Distance(4,3,2,1)` returns 2.82 (approximately)



Halt

Syntax: Halt( )

Description: Terminates the simulation at the next safe stopping place and sends simulation completion information to the Output window. Also see “Kill” on page 257.

Example: `Model.Halt( )`

Kill

Syntax: Kill( )

Description: Terminates the model execution immediately. Also see “Halt” on page 257.

Example: `Model.Kill( )`

Message

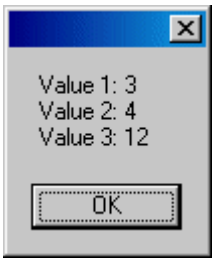
Syntax: Message(*expression to evaluate*, *value 1*, *value 2*, *value 3*)

Message(*value*)

Description: Displays a dialog box with the specified information.

Example: `Model.Message(i==3,x,y,z)` displays a dialog box with the values for x, y, and z if i is equal to 3.

`Model.Message ("I'm Here")` displays a dialog box with the phrase "I'm Here".



Example of message dialog box showing value 1, value 2, and value 3.

Pause

Syntax: Pause( )

Description: Pauses the model execution. This function is useful for debugging models. Place the Pause function in a task before a known error and then run the model. The model executes in a single-step fashion immediately before the error occurs.

Example: `Model.Pause( )`

Print Output

Syntax: PrintOutput(*value*)

Description: Writes a message to the Output window.

Example: `Model.PrintOutput("In task 2 Beginning effect")`

Random

Syntax: Random(*minimum*, *maximum*)

Random( )

Description: Returns a random number between the two numbers you supply or returns a random number between 0 and 1, including 0 and excluding 1.

Example: `Model.Random(2,4)` returns a random number between the numbers 2 and 4.

`Model.Random( )` returns a random number between 0 and 1, including 0 and excluding 1.

Random Integer

Syntax: RandomInteger(*minimum*, *maximum*)

Description: Returns a random integer between the minimum and maximum numbers that you supply.

Example: `Model.RandomInteger(1,4)` returns a random integer between the numbers 1 and 4.

Random Seed

Syntax: RandomSeed

Description: Returns the value of the current random number seed. Do not use parentheses after `RandomSeed`.

Example: In the example `MySeed = Model.RandomSeed` the variable `MySeed` would contain the current seed. The same model run with this seed will produce the same results.

Resume

Syntax: Resume(“*key”, value*)

Description: Resumes all entities matched by the key-value pair. The key parameter can be a default entity attribute or a user-defined attribute.

Example: `Model.Resume (“Tag”, 3)` resumes any entities with a tag value of 3.

`Model.Resume(“ID”, 2)` resumes any entities in task ID 2.

Resume (array)

Syntax: Resume("*key array", value array*)

Description: Resumes all entities matched by the key-value pair. The key parameter can be a default entity attribute or a user-defined attribute.

Example: If key[0] = "Group," key[1] = "Tag," val[0] = 3, and val[1] = 12, `Model.Resume(key, val)` resumes any entities with a group value of 3 and a tag value of 12.

If key[0] = "Group," key[1] = "ID," val[0] = 3, and val[1] = 12, then `Model.Resume (key,val)` resumes any entities with a group value of 3 in task ID 12.

Run Number

Syntax: RunNumber

Description: Returns the current run number. Do not use parentheses after RunNumber.

Example: In the example `MyRun = Model.RunNumber` the variable `MyRun` will store the current run number. This is useful for snapshots of multiple runs.

Send External Variable

Syntax: SendExternalVariable (*variable name*)

Description: Sends a variable to an external connected program. This only occurs if Micro Saint Sharp is started or "owned" by another application.

Example: `Model.SendExternalVariable(MyRun)`

Start

Syntax: Start(ID, tag number)

Start(Entity Variable)

Description: Starts the entity specified by the tag number in the specified task.

Example: `Model.Start (2,2)` starts the entity with the tag value 2 in task ID 2.

```
TempEnt=Model.CreateEntity( );
TempEnt.Tag = 3; TempEnt.ID = 2;
Start(TempEnt);
```

starts the entity with the tag value 3 in task ID 2.

Stop

Syntax: Stop("*key", value*)

Description: Stops all entities matched by the key-value pair. The key parameter can be a default entity attribute or a user-defined attribute.

Example: `Model.Stop ("Tag", 3)` stops any entities with a tag value of 3.

`Model.Stop("ID", 2)` stops any entities in task ID 2.

Stop (array)

Syntax: Stop("*key array", "value array"*)

Description: Stops all entities matched by the key-value pair. The key parameter can be a default entity attribute or a user-defined attribute.

Example: If key[0] = "Group," key[1] = "Tag," val[0] = 3, and val[1] = 12, then `Model.Stop(key, val)` suspends any entities with a group value of 3 and a tag value of 12.

If key[0] = "Group," key[1] = "ID," val[0] = 3, and val[1] = 12, then `Model.Stop (key,val)` stops any entities with a group value of 3 in task ID 12.

Suspend

Syntax: suspend("*key", value*)

Description: Suspends all entities matched by the key-value pair. The key parameter can be a default entity attribute or a user-defined attribute.

Example: `Model.Suspend ("Tag", 3)` suspends any entities with a tag value of 3.

`Model.Suspend("ID", 2)` suspends any entities in task ID 2.

Suspend (array)

Syntax: Suspend("*key array", value array*)

Description: Suspends all entities matched by the key-value pair. The key parameter can be a default entity attribute or a user-defined attribute.

Example: If key[0] = "Group," key[1] = "Tag," val[0] = 3, and val[1] = 12, then `Model.Suspend(key, val)` suspends any entities with a group value of 3 and a tag value of 12.

If key[0] = "Group," key[1] = "ID," val[0] = 3, and val[1] = 12 then `Model.Suspend (key,val)` suspends any entities with a group value of 3 in task ID 12.

To String

Syntax: ToString( )

Description: Converts a value to a string.

Example: `Model.ToString( )`

Trigger Snapshot

Syntax: TriggerSnapshot(*snapshot ID*)

Description: This function executes the snapshot with the indicated ID (string value). The function returns true if the snapshot was executed and returns false if the snapshot was not executed. This function is an alternate way to make snapshots occur if the default triggers do not fulfill your needs.

Example: `Model.TriggerSnapshot(Snap1)`

Wall Clock

Syntax: WallClock(*type*)

Description: Returns the number of seconds or minutes since the program started running. Use the value 0 for seconds and use the value 1 for minutes.

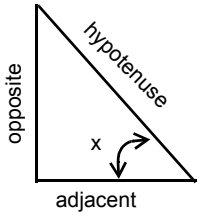
Example: `Model.WallClock(1)` returns the number of minutes since the model started running.

Math Functions

Mathematical functions are specific to mathematical operations, such as minimum, maximum, and trigonometric functions. All Math functions start with the word Math, followed by a period, and then followed by the function name, such as `Math.Cos( )`.

✓ Note:
Function names are case-sensitive.

The following diagram shows the parameters used in trigonometric functions.



Constants

The Mathematical class of functions contains two constants:

- E represents the natural logarithmic base specified by the constant e. You call this constant with the phrase `Math.E`.
- Pi represents the ratio of the circumference of a circle to its diameter and is specified by the constant Π. You call this constant with the phrase `Math.Pi`.

Abs

Syntax: Abs(*value*)

Description: Returns the absolute value of (value).

Example: `Math.Abs(3)` returns the value 3.

Acos

Syntax: Acos(*value*)

Description: Returns the angle, in degrees (0-180), whose cosine is (value).

Example: `Math.Acos(adj/hyp)` returns x.

Asin

Syntax: Asin(*value*)

Description: Returns the angle, in degrees (90-270), whose sine is (value).

Example: `Math.Asin(opp/hyp)` returns x.

Atan

Syntax: Atan(*value*)

Description: Returns the angle, in degrees (90-270), whose tangent is (*value*).

Example: `Math.Atan(opp/adj)` returns x.

Atan2

Syntax: Atan2(*value 1, value 2*)

Description: Returns the angle whose tangent is the quotient of two specified numbers.

Example: `Math.Atan2(.5,1)` returns the value 2.21429744.

BigMul

Syntax: BigMul(*int a, int b*)
where a and b are 32-bit integers

Description: Returns the multiplication of a and b as a long integer (64-bit).

Example: `BigMul(2, 3)` returns 6.

Ceiling

Syntax: Ceiling(*value*)

Description: Returns the largest whole number greater than or equal to the specified number.

Example: `Math.Ceiling(5.2)` returns 5.

Cos

Syntax: Cos(*value*)

Description: Returns the cosine of the angle value you supply. Enter the angle in radians. To convert degrees to radians, multiple by Π/180.

Example: `Math.cos(x)` returns the quotient adj/hyp.

Cosh

Syntax: Cosh(*value)*

Description: Returns the hyperbolic cosine of the specified angle. Enter the angle in radians. To convert degrees to radians, multiple by Π/180.

Example: `Math.Cosh(x)` returns the quotient adj/hyp.

DivRem

Syntax: DivRem(*value1*, *value 2*)

Description: Division remainder. Returns the remainder from the division operation. Only integers can be used. For floating point division, see “IEEERemainder” on page 269.

Example: `Math.DivRem(3,2)` returns the value 1.

Exp

Syntax: Exp(*value*)

Description: Returns e raised to the specified power.

Example: `Math.Exp(2)` returns the value 7.3890561.

Floor

Syntax: Floor(*value*)

Description: Returns the smallest whole number less than or equal to the specified number.

Example: `Math.Floor(3.8)` returns the value 3.

IEEERemainder

Syntax: IEEERemainder(dividend, divisor)

Description: Floating point division remainder (can be used to calculate division error).

Example: `Math.IEEERemainder(1.797693e+308,3.00)` returns `-1.995840e+292`.

Log

Syntax: log(*value*)

Description: Returns the natural logarithm of a specified number. *Value* must be greater than zero.

Example: `Math.Log(10)` returns the value 1.

Log10

Syntax: log10(*value*)

Description: Returns the base 10 logarithm of a specified number. *Value* must be greater than zero.

Example: `Math.Log10(10^7)` returns the value 7.

Max

Syntax: Max(*value1*, *value2*)

Description: Returns the maximum of the two values you supply.

Example: `Math.Max(1,2)` returns the value 2.

Min

Syntax: Min (*value1*, *value2*)

Description: Returns the minimum of the two values you supply.

Example: `Math.Min(1,2)` returns the value 1.

Pow

Syntax: Pow (*value1*, *value2*)

Description: Returns the value of a specified number raised to the specified power.

Example: `Math.Pow(2,3)` returns the value 8.

Round

Syntax: Round (*value1*)

Description: Returns the number nearest to the specified value. Rounds value up or down to the nearest whole number. If fractional part is >= 0.5, value is rounded up (away from 0). Otherwise, it is rounded down (toward 0).

Example: `Math.Round(2.5)` returns the value 2.

Sign

Syntax: Sign(*value*)

Description: Returns a value indicating the sign of the number. If the value is less than zero, -1 is returned. If the value is equal to zero, 0 is returned. If the value is greater than zero, a 1 is returned.

Example: `Math.Sign(943839)` returns the value 1.

Sin

Syntax: Sin(*value*)

Description: Returns the sine of the angle you supply. Enter the angle in radians. To convert degrees to radians, multiply by Π/180.

Example: `Math.Sin(1.2)` returns the value 0.932039086.

Sinh

Syntax: Sinh(*value*)

Description: Returns the hyperbolic sine of specified angle. Enter the angle in radians. To convert degrees to radians, multiply by Π/180.

Example: `Math.Sinh(1.2)` returns the value 1.50946136.

Sqrt

Syntax: Sqrt(*value*)

Description: Returns the square root of the specified number.

Example: `Math.Sqrt(4)` returns the value 2.

Tan

Syntax: Tan(*value*)

Description: Returns the tangent of the angle you supply. Enter the angle in radians. To convert degrees to radians, multiply by Π/180.

Example: `Math.Tan(1.2)` returns the value 2.57215162.

Tanh

Syntax: Tanh(*value*)

Description: Returns the hyperbolic tangent of the specified angle. Enter the angle in radians. To convert degrees to radians, multiply by Π/180.

Example: `Math.Tanh(1.2)` returns the value 0.833654607.

Distribution Functions

Distribution functions are specific to probability functions. Micro Saint Sharp contains 20 distribution functions that return a random number from a probability distribution. All Distribution functions start with the word Distributions, followed by a period, and then followed by the function name, such as `Distributions.Beta`.

✓ Note:
Function names are case-sensitive.

Additional details on distribution functions are provided in Chapter 13: "Probability Distributions." For specific details regarding the probability distributions, consult a statistics text.

Distribution function parameters include the following:

- Psuccess. The probability of success,
- Successes. The number of successes.
- Mean. The mean value.
- Mode. The most common or frequent value.
- Standard deviation (square root of the arithmetic average of the squares of the deviations from the mean).
- Ntrials. The number of trials.
- Minimum. The minimum value to be used in the distribution.
- Maximum. The maximum value to be used in the distribution.
- Scale. The scale of measurement of the values in the range of distribution (for more details, see "Overview of Probability Distributions" on page 300).
- Shape, Shape 1, Shape 2. The basic form or shape of a distribution within the general family of distributions of interest (for more details, see "Overview of Probability Distributions" on page 300).

Bernoulli

Syntax: Bernoulli(*psuccess*)

Description: Returns a random integer from a Bernoulli distribution. You supply the probability of success (*psuccess*) to define the distribution. The value you supply for *psuccess* should be greater than or equal to zero and less than 1.

Example: `Distributions.Bernoulli(.1)`

Beta

Syntax: Beta(*mean*, *standard deviation*)

Description: Returns a random number from a Beta distribution. You supply the mean and standard deviation to define the distribution.

Example: `Distributions.Beta(4,1)`

Binomial

Syntax: Binomial(*ntrials*, *psuccess*)

Description: Returns a random integer from a binomial distribution. You supply the number of trials (*ntrials*) and the probability of success (*psuccess*) values to define the distribution. The number returned represents the number of successes that occur during the trials; this number can be any integer between and including zero and *ntrials*. The value you supply for *psuccess* should be greater than or equal to zero and less than 1.

Example: `Distributions.Binomial(5,.1)`

DiscreteUniform

Syntax: DiscreteUniform(*minimum*, *maximum*)

Description: Returns a random integer from a Discrete Uniform distribution. You supply the minimum and maximum values to define the distribution.

Example: `Distributions.DiscreteUniform(4,1)`

Exponential

Syntax: Exponential(*mean*)

Description: Returns a random number from an exponential distribution. You supply the mean value to define the distribution. A standard deviation is not used for this function.

Example: `Distributions.Exponential(4)`

Extreme Value Type A

Syntax: ExtremeValueTypeA(*mean*, *standard deviation*)

Description: Returns a random number from an Extreme Value Type A distribution. You supply the mean and standard deviation to define the distribution.

Example: `Distributions.ExtremeValueTypeA(4,1)`

Extreme Value Type B

Syntax: ExtremeValueTypeB(*mean*, *standard deviation*)

Description: Returns a random number from an Extreme Value Type B distribution. You supply the mean and standard deviation to define the distribution.

Example: `Distributions.ExtremeValueTypeB(4,1)`

Gamma

Syntax: Gamma(*mean*, *standard deviation*)
Gamma(*shape*)

Description: Returns a random number from the Gamma distribution, using either the mean and standard deviation or the shape factor.

Example:
```
Distributions.Gamma(4,1)
Distrubution.Gamma(4)
```

Geometric

Syntax: Geometric(*mean*)

Description: Returns a random number from a Geometric distribution. You supply the mean to define the distribution.

Example:
```
Distributions.Geometic(4)
```

Inverse Gaussian

Syntax: InverseGaussian(*mean*, *standard deviation*)

Description: Returns a random number from an Inverse Gaussian distribution. You supply the mean and standard deviation to define the distribution.

Example:
```
Distributions.InverseGaussian(4,1)
```

Logistic

Syntax: Logistic(*mean*, *standard deviation*)

Description: Returns a random number from a Logistic distribution. You supply the mean and standard deviation to define the distribution.

Example: `Distributions.Logistic(4,1)`

LogLogistic

Syntax: LogLogistic(*scale*, *shape*)

Description: Returns a random number from a LogLogistic distribution. You supply the scale and shape factors to define the distribution.

Example: `Distributions.LogLogistic(4,1)`

Lognormal

Syntax: Lognormal(*mean*, *standard deviation*)

Description: Returns a random number from a lognormal distribution. You supply the mean and standard deviation values to define the distribution.

Example: `Distributions.Lognormal(4,1)`

Negative Binomial

Syntax: NegativeBinomial(*successes*, *psuccess*)

Description: Returns a random number (integer) from a Negative Binomial distribution. You supply the number of successes value and the probability of success to define the distribution.

Example: `Distributions.NegativeBinomial(4,1)`

Normal

Syntax: Normal(*mean*, *standard deviation*)

Description: Returns a random number from a normal distribution. You supply the mean and standard deviation values to define the distribution.

Example: `Distributions.Normal(4,1)`

Pareto

Syntax: Pareto(*minimum*, *scale*)

Description: Returns a random number from a Pareto distribution. You supply the minimum value scale to define the distribution.

Example: `Distributions.Pareto(4,1)`

Pearson Type 5

Syntax: PearsonType5(*scale*, *shape*)

Description: Returns a random number from a Pearson Type 5 distribution. You supply the scale and shape factors to define the distribution.

Example: `Distributions.PearsonType5(4,1)`

Pearson Type 6

Syntax: PearsonType6(*scale*, *shape1*, *shape2*)

Description: Returns a random number from a Pearson Type 6 distribution. You supply the scale and shape factors to define the distribution.

Example: `Distributions.PearsonType6(3, 4,1)`

Poisson

Syntax: Poisson(*mean*)

Description: Returns a random number from a poisson distribution. The number returned represents the number of independent events that occur in a particular time interval. The mean value you supply defines the average number of events expected in such time intervals.

Example: `Distributions.Poisson(4)`

Rectangular

Syntax: Rectangular*(mean, minimum*)

Description: Returns a random number from the Rectangular distribution. You supply the mean and minimum values to define the distribution.

Example: `Distributions.Rectangular(4,1)`

Triangular

Syntax: Triangular(*mode*, *minimum*, *maximum*)

Description: Returns a random number from a Triangular distribution. You supply the mode, minimum, and maximum values to define the distribution.

Example: `Distributions.Triangular(4,1, 9)`

Weibull

Syntax: Weibull(*scale*, *shape*)

Description: Returns a random number from a Weibull distribution. You supply the scale and shape factors to define the distribution.

Example: `Distributions.Weibull(4,1)`

Animator Functions

Animator functions are specific to Micro Saint Sharp Animator (Animator is an optional add-on component). All Animator functions start with the word Animator, followed by a period, and then followed by the function name, such as `Animator.GetImage(2)`.

> **Note:**
> Function names are case-sensitive.

Variable types for Animator functions are listed in the syntax for a function and include int for integer, float for floating, object for objects, and IEnumerable for storage variables that can be enumerated.

IEnumerable is a Microsoft C# Sharp interface for a special type of data container, such as an array, an array list, a hash table (tables containing associations between keys and values so that given a key the value can be found quickly), or any other container that implements the IEnumerable interface. For details on variable types, see page 118.

Animator parameters include the following:

- ImageID. The Object ID for the image. The Object ID displays in the Properties window for the image.
- ImageIDs. An IEnumerable containing the Image IDs.
- TypeID. The ID shared by all copies of a particular palette imageID. The TypeID displays in the Properties window for the image.
- ObjectID. The unique ID for the image assigned by Micro Saint Sharp. The Object ID displays in the Properties window for the image in the field UniqueID.
- WayPointID. The unique ID for the way point. The WayPointID displays in the Properties window for the image in the field UniqueID.

- CommentID. The unique ID for the comment. The Comment ID displays in the Properties window for the image in the field UniqueID.
- StartID. The starting way point ID. The StartID displays in the Properties window for the image in the field UniqueID.
- EndID. The ending way point ID. The EndID displays in the Properties window for the image in the field UniqueID.
- Duration. The amount of simulation time.
- Color. The color.
- X. The x-coordinate position on the Animator display. The (0,0) position in the Animator display is the upper left corner of the window. To determine the coordinates of a position, click anywhere on the Animator display and the coordinates display in the status bar.
- Y. The y-coordinate position on the Animator display. The (0,0) position in the Animator display is the upper left corner of the window. To determine the coordinates of a position, click anywhere on the Animator display and the coordinates display in the status bar.
- PointList. An IEnumerable containing PointF objects. A PointF object is basically a coordinate in decimal form.

  As an example, assume you use the following to create a PointF object:

  ```
  PointF pt1 = new PointF(1.0f, 4.9f);
  ```

  To add this to an array List, you could do the following:

  ```
  PointF pt1 = new PointF(1.0f, 4.9f);
  ArrayList ptList = new ArrayList();
  ptList.Add(pt1);
  ```

  You can then pass `ptList` as an array of points.

To add this to an hash table, you could do the following:

```
PointF pt1 = new PointF(1.0f, 4.9f);
Hashtable ptHash = new Hashtable ();
ptHash.Add(pt1);
```

You can then pass `ptHash` as an IEnumerable of points.

To add this to an array, you could do the following:

```
PointF pt1 = new PointF(1.0f, 4.9f);
Object[] ptArray = new Object[1];
ptArray[0] = pt1;
```

You can then pass `ptArray` as an IEnumerable of points.

- WayPointIDList. An IEnumerable containing the Unique IDs for way points.
- Angle. Any angle, negative or positive. For example, -500, 180, and 1000 are all valid values.
- Width. The width of the image in pixels.
- Height. The height of the image in pixels.
- Text. A string of text.
- Rft. An instance of the RotateFlipType enum.

ClearBackground

Syntax: ClearBackground( )

Description: Clears the background of the Animator display.

Example: `Animator.ClearBackground( )` clears the Animator display background.

ColorFilter

Syntax: ColorFilter(object imageID, int red, int green, int blue)

ColorFilter(object imageID, Color color)

Description: Applies a color filter to an image using the specified hues of red, green, and blue or the specified color.

Example: `Animator.ColorFilter(3, 0, 0, 0)` changes the image with ID 3 to black (total absence of color).

`Animator.ColorFilter(3, "Black")` colors the image with ID 3 to black.

CopyImage

Syntax: CopyImage(object imageID, float x, float y)

CopyImage(object imageID, copied object ID, float x, float y)

Description: Copies an image with the specified ID.

CopyImage(object imageID, float x, float y) copies the image, sets the copied image's ID, and places the image at the assigned x, y coordinates.

CopyImage(object imageID, copied object ID, float x, float y) copies the image, assigns it the specified ID, and places the image at the assigned x, y coordinates.

Example: `Animator.CopyImage(2, 10, 20)` copies the image with ID 2 and places it at the coordinate (10, 20).

`Animator.CopyImage(2, 3, 10, 20)` copies the image with ID 2, assigns it to image ID 3, and places it at the coordinate (10, 20).

CreateComment

Syntax: CreateComment(float x, float y)

CreateComment(float x, float y, object text)

Description: Creates a comment in the Animator display. Returns the assigned comment ID.

Example: `x = Animator.CreateComment(10, 20)` creates a comment at the coordinates (10, 20); x will be equal to the created comment's new ID value.

`x = Animator.CreateComment(10, 20, "Here's a comment")` creates a comment at the coordinates (10, 20) with the text "Here's a comment"; x will be equal to the created comment's new ID value.

CreateImage

Syntax: CreateImage(object imageTypeID, object objectID, float x, float y)

CreateImage(object imageTypeID, float x, float y)

Description: Creates an image in the Animator display using one of the images in the Animator palette.

CreateImage(object imageTypeID, object objectID, float x, float y) creates an image in the Animator display, assigns the image an ID, and places the image at the specified x, y coordinates.

CreateImage(object imageTypeID, float x, float y) creates an image in the Animator display, assigns the image an ID, places the image at the specified x,y coordinates, and returns the assigned image ID value.

Example: `Animator.CreateImage(2, 3, 10, 20)` creates an image in the Animator display using the image in the palette with an ID of 2, assigns the image an ID of 3, and places the image at the coordinates (10, 20).

x = `Animator.CreateImage(2, 10, 20)` creates an image in the Animator display using the image in the palette with an ID of 2 and places the image at the coordinates (10, 20); x will be equal to the created image's new ID value.

CreateImageAtWayPoint

Syntax: CreateImageAtWayPoint (object imageTypeID, object objectID, object wayPointID)

CreateImageAtWayPoint (object imageTypeID, object wayPointID)

Description: Creates an image in the Animator display using the images in the Animator palette.

CreateImageAtWayPoint (object imageTypeID, object objectID, object wayPointID) creates an image in the Animator display, assigns the image an ID, and places the image at the specified way point.

CreateImageAtWayPoint (object imageTypeID, object wayPointID) creates an image in the Animator display and places the image at the specified way point.

Example: `Animator.CreateImage(2, 3, 4)` creates an image in the Animator display using the image in the palette with an ID of 2, assigns the image an ID of 3, and places the image at way point 4.

`x = Animator.CreateImage(2, 4)` creates an image in the Animator display using the image in the palette with an ID of 2 and places the image at way point 4; x will be equal to the created image's new ID value.

CreateWayPoint

Syntax: CreateWayPoint(float x, float y)

Description: Creates a way point in the Animator display. Returns the assigned way point ID.

Example: `x = Animator.CreateWayPoint(10, 20)` creates a way point at the coordinates (10, 20); x will be equal to the created way point's new ID value.

FollowPath

Syntax: FollowPath(object imageID, object startID, object endID, float duration)

Description: Moves an image from one way point to another.

Example: `Animation.FollowPath(3, 4, 5, 6)` moves the image with ID 3 from way point 4 to way point 5 over the period of six time units.

GetImage

Syntax: GetImage(object imageID)

Description: Returns an animator object containing the properties of a user-defined image.

Example: `x = Animator.GetImage(2)` x is a copy of the image, so `x.Height` equals the image's height (in pixels); x must be defined as an animator object variable type.

`x = Animator.GetImage(2). Height` returns the height (in pixels) of the animator image with ID 2.

GetWayPoint

Syntax: GetWayPoint(object wayPointID)

Description: Returns an animator object containing the properties of a user-defined way point.

Example: `x = Animator.GetWaypoint(2)` x is a copy of the way point, so `x.CenterX` is the x location of the way point; x must be defined as an animator object variable type.

`x = Animator.GetWayPoint(2).CenterX` returns the x location of way point 2.

HideImage

Syntax: HideImage(object imageID)

Description: Hides an image in the Animator display. To redisplay the image, see “ShowImage” on page 296.

Example: `Animator.HideImage(3)` hides the image with ID 3.

HideImages

Syntax: HideImages(IEnumerable imageIDs)

Description: Hides a series of images in the Animator display. To redisplay the image, see “ShowImages” on page 297.

Example: `Animator.HideImages(x)` hides all the images in the variable list x.

HideComment

Syntax: HideComment(object commentID)

Description: Hides a comment in the Animator display. To redisplay the image, see “ShowComment” on page 296.

Example: `Animator.HideComment(3)` hides the comment with ID 3 in the Animator display.

HideComments

Syntax: HideComments(IEnumerable imageIDs)

Description: Hides a series of comments from the Animator display. To redisplay the image, see “ShowComments” on page 296.

Example: `Animator.HideComments(x)` hides all the comments in the variable list x.

MoveComment

Syntax: MoveComment(object commentID, float x, float y)

Description: Moves a comment box to a new location.

Example: `Animator.MoveComment(3, 10, 20)` moves the comment with the ID 3 to the coordinates (10, 20).

MoveImage

Syntax: MoveImage(object imageID, float x, float y, float duration)

MoveImage(object imageID, object wayPointID, float duration)

Description: Moves an image to a specified x, y coordinate or to a specified way point.

Example: `MoveImage(3, 10, 20, 5)` moves the image with the ID 3 to the coordinates (`10, 20`) over a period of five time units (default is seconds if the model is set to run in real time).

`MoveImage(3, 4, 5)` moves the image with the ID 3 to way point 4 over a period of five time units (the default is seconds if the model is set to run in real time).

MoveImageDown

Syntax: MoveImageDown(object imageID)

Description: Moves an image down one layer. All images have a "Z" property, which is the order that images are on top of each other. Images with a lower "Z" property are closer to the bottom.

Example: `Animator.MoveImageDown(2)` moves image 2 down one layer.

MoveImageUp

Syntax: MoveImageUp(object imageID)

Description: Moves an image up one layer. All images have a "Z" property, which is the order that images are on top of each other. Images with a higher "Z" property are closer to the top.

Example: `Animator.MoveImageUp(2)` moves image 2 up one layer.

MoveImageToBottom

Syntax: MoveImageToBottom(object imageID)

Description: Moves an image to the bottom of the view. If two images are on top of each other, the one on the bottom is covered.

Example: `Animator.MoveImageToBottom(2)` places the image with ID 2 on the bottom of any other image it passes over.

MoveImageToPoints

Syntax: MoveImageToWayPoints(object imageID, IEnumerable pointList, float duration)

Description: Moves an image along a series of specified coordinates.

Example: `Animator.MoveImageToWayPoints(3, x, Entity.Duration)` moves the image with ID 3 through the list variable x (containing a series of coordinates) over the current entities duration.

MoveImageToTop

Syntax: MoveImageToTop(object imageID)

Description: Moves an image to the top of the view. If two images are on top of each other, the one on top covers the one on bottom.

Example: `Animator.MoveImageToTop(2)` places the image with ID 2 on top of any other image it passes over.

MoveImageToWayPoints

Syntax: MoveImageToWayPoints(object imageID, IEnumerable wayPointIDList, float duration)

Description: Moves an image along a series of specified way points.

Example: `Animator.MoveImageToWayPoints(3, x, Entity.Duration)` moves the image with ID 3 through the list variable x (containing a series of way points) over the current entities duration.

PlaceImage

Syntax: PlaceImage(object imageID, float x, float y)

PlaceImage(object imageID, object wayPointID)

Description: Jumps an object to the specified coordinate or way point in the Animator display.

Example: `Animator.PlaceImage(3, 10, 20)` jumps the image with ID 3 to the coordinates (10, 20).

`Animator.PlaceImage(3, 4)` jumps the image with ID 3 to way point 4.

RemoveImage

Syntax: RemoveImage(object imageID)

Description: Removes an image from the Animator display.

Example: `Animator.RemoveImage(3)` removes the image with ID 3 from the Animator display.

RemoveImages

Syntax: RemoveImages(IEnumerable imageIDs)

Description: Removes a series of images from the Animator display.

Example: `Animator.RemoveImages(x)` removes all the images in the variable list x.

RemoveComment

Syntax: RemoveComment(object commentID)

Description: Removes a comment from the Animator display.

Example: `Animator.RemoveComment(3)` removes the comment with ID 3 from the Animator display.

RemoveComments

Syntax: RemoveComments(IEnumerable imageIDs)

Description: Removes a series of comments from the Animator display.

Example: `Animator.RemoveComments(x)` removes all the comments in the variable list x.

RotateImage

Syntax: RotateImage(object imageID, float angle)

RotateImage(object imageID, float angle, float duration)

Description: Rotates an image to a specified angle or to a specified angle over a period of time.

Example: `RotateImage(3, 45)` rotates the image with ID 3 to an angle of 45 degrees.

`RotateImage(3, 45, 5)` rotates the image with ID 3 to an angle of 45 degrees for the duration of five time units.

RotateFlipImage

Syntax: RotateFlipImage(object imageID, RotateFlipType rft)

Description: Rotates/flips an image. You specify the direction of the image rotation and the axis used to flip the image.

Example: `Animator.RotateFlipImage 3,RotateFlipType.Flip)` flips image 3.

ScaleImage

Syntax: ScaleImage(object imageID, int width, int height)

Description: Scales an image.

Example: `Animator.ScaleImage(3, 10, 10)` scales the image with ID 3 to a height of 10 pixels and a width of 10 pixels.

SetBackground

Syntax: SetBackground(object imageTypeID)

Description: Sets the background of the Animator display.

Example: `Animator.SetBackground(2)` sets the background of the Animator display to the image on the palette with the ID 2.

SetBottomImageText

Syntax: SetBottomImageText(object imageID, object text)

Description: Sets an object bottom image text.

Example: `Animator.SetBottomImageText(3, "Ball")` places the text "Ball" below the image with ID 3.

SetCommentText

Syntax: SetCommentText(object commentID, object text)

Description: Sets the text of a comment on the Animator display.

Example: `Animator.SetCommentText(3, "Changed Text")` changes the comment with ID 3 to the text "Changed Text."

SetTopImageText

Syntax: SetTopImageText(object imageID, object text)

Description: Set the top image text for an object.

Example: `Animator.SettopImageText(3, "Ball")` places the text "Ball" above the image with ID 3.

ShowComment

Syntax: ShowComment(object commentID)

Description: Shows a comment that was hidden in the Animator display (see “HideComment” on page 289).

Example: `Animator.ShowComment(3)` shows the comment with ID 3 from the Animator display.

ShowComments

Syntax: ShowComments(IEnumerable imageIDs)

Description: Shows a series of comments that were hidden in the Animator display (see “HideComments” on page 289).

Example: `Animator.ShowComments(x)` shows all the comments in the variable list x.

ShowImage

Syntax: ShowImage(object imageID)

Description: Shows an image that was hidden on the Animator display (see “HideImage” on page 289).

Example: `Animator.ShowImage(3)` shows the image with ID 3.

ShowImages

Syntax: ShowImages(IEnumerable imageIDs)

Description: Shows a series of images that were hidden on the Animator display (see “HideImages” on page 289).

Example: `Animator.ShowImages(x)` shows all the images in the variable list x.

■ Animator Functions

13

Probability Distributions

To help simulate task execution times as realistically as possible, Micro Saint Sharp randomly generates the execution times for each task using a probability distribution. When you select a time distribution for a task, Micro Saint Sharp uses the distribution to generate random execution times that occur in the pattern predicted by the distribution.

A probability distribution defines how frequently a particular value is likely to occur in a set of observations. For example, in a Normal distribution, values close to the mean are likely to occur, while values far from the mean are not. If you define task times to be distributed normally with a mean of 40 minutes and a standard deviation of 5, the task would usually execute with a time between 30 and 50 minutes.

Micro Saint Sharp provides 20 basic and advanced probability distributions for your use.

Overview of Probability Distributions

Probability distributions can be discrete or continuous.

- *Discrete* distributions return integer values and are primarily used in simulating numbers of random events and not in simulating times. Discrete probability distributions provided by Micro Saint Sharp include Bernoulli, Binomial, Discrete Uniform, Geometric, Negative Binomial, and Poisson.

- *Continuous* distributions include mean/standard deviation distributions, scale/shape distributions, and minimum value distributions.

  Mean/standard deviation distributions use mean and/or standard deviation as inputs for the distribution. Mean/standard continuous probability distributions provided by Micro Saint Sharp include Beta, Exponential, Extreme Value Type A, Extreme Value Type B, Gamma, Inverse Gaussian, Logistic, Lognormal, and Normal distributions.

  Scale/shape distributions use standard scale and shape parameters as inputs for the distribution (see the following page for a description of these parameters). These distributions can have undefined means and standard deviations or are not trivial to convert to a mean and standard deviation-dependent function. Scale/shape continuous probability distributions provided by Micro Saint Sharp include LogLogistic, Pearson Type 5, Pearson Type 6, and Weibull.

  Minimum value distributions use a minimum and/or maximum in addition to mean and/or scale as inputs for the distribution. Minimum value continuous probability distributions provided by Micro Saint Sharp include the Pareto, Rectangular, and Triangular.

The equations defining the probability distributions use several parameters. In addition to the more common statistical variables such as mean and standard deviation, the following parameters are used:

- A *scale* parameter, determines the scale of measurement of the values in the range of distribution. A change in the scale parameter compresses or expands the distribution without altering its basic form.

- A *shape* parameter, determines the basic form or shape of a distribution within the general family of distributions of interest. A change in the shape parameter generally alters the properties of a distribution, such as skewness, more than a change in location or scale. Some distributions do not have a shape parameter, while others may have several.

References for Probability Distributions

The following section describes the probability distributions provided by Micro Saint Sharp. The descriptions are used with the permission of Stat::Fit and are modified from the following sources:

- Johnson, N.L., Kotz, Samuel, Balakrishnan, 1994, "Continuous Univariate Distributions, Volume 1": John Wiley & Sons.

- Johnson, N.L., Kotz, Samuel, Kemp, A.W., 1992, Univariate Discrete Distributions: John Wiley & Sons.

- Law, A.M., and Kelton, W.D., 1991, Simulation Modeling & Analysis: McGraw-Hill.

- Shooman, M.L., 1990, Probabilistic Reliability: An Engineering Approach.

Types of Probability Distributions

Micro Saint Sharp provides 20 basic and advanced probability distributions for your use.

Bernoulli Distribution

Syntax: Bernoulli(*psuccess*)

Applications: The Bernoulli distribution models the number of successes that can occur in a single trial with probability of success *psuccess*. This distribution best describes situations where a "trial" is made, resulting in either success or failure. Examples include tossing a coin or modelling the success or failure of a procedure. A Bernoulli trial is an experiment in which only two outcomes are possible: success, with probability p, and failure, with probability 1-p.

Description: The Bernoulli distribution is a discrete distribution that is bounded by [0, n]. The Bernoulli distribution is a Binomial distribution with the number of trials = 1.

Beta Distribution

Syntax: Beta (*mean, standard deviation*)

Applications: The Beta distribution provides distributions of random proportion and is used for models in the absence of data. Examples include the proportion of defective items in a shipment, the time to complete a task in a network, dissipation rate in breakage models, construction duration, and tool wear.

Description: The Beta distribution is a continuous distribution that has both upper and lower finite bounds. Because many real situations can be bounded in this way, the Beta distribution can be used empirically to estimate the actual distribution before much data are available.

Binomial Distribution

Syntax: Binomial(*ntrials, psuccess*)

Applications: The Binomial distribution models the number of successes that can occur in a sequence of *ntrials* with probability of *psuccess* in each trial. For example, if you test a product part five times and each test has a ten percent probability of success, the number of successes is usually zero or one. If each test has a fifty percent probability of success, the number of successes would usually be two or three.

The Binomial distribution is used extensively in games, but is also useful in genetics, sampling of defective parts in a stable process, and other event sampling tests where the probability of the event is known to be constant or nearly constant.

Description: The Binomial distribution is a discrete distribution that is bounded by [0, n]. Typically, it is used where a single trial is repeated over and over.

Discrete Uniform Distribution

Syntax: DiscreteUniform(*minimum, maximum*)

Applications: The Discrete Uniform distribution is used where there is random occurrence with several possible outcomes, each of which is equally likely.

Description: The Discrete Uniform distribution is a discrete distribution that is bounded on [*minimum*, *maximum*] with constant probability at every value on or between the bounds. It arises when an event can have a finite and equally probable number of outcomes.

Exponential Distribution

Syntax: Exponential(*mean*)

Applications: The Exponential distribution is frequently used to represent the time between random occurrences. Examples include the time between arrivals at a specific location in a queuing model or the time between failures in reliability models. The Exponential distribution is also used to represent times between independent events that occur at a constant rate, such as the time between arrival of aircraft at airports, between arrival of patients at a hospital, or between accidents at a factory. Typically, Exponential distributions are used for arrival rates.

Description: The Exponential distribution is a continuous distribution that is bounded on the lower side. Its shape is always the same, starting at a finite value at the minimum and continuously decreasing at larger x. An Exponential distribution does not have a standard deviation; the distribution is defined by the mean value. The most frequently occurring values are close to zero. As the value increases, the number of occurrences decreases.

Extreme Value Type A

Syntax: ExtremeValueTypeA(*mean*, *standard deviation*)

Applications: The Extreme Value Type A distribution describes the limiting distribution of the extreme values of many types of samples. Examples include parameters in growth models, astronomy, human lifetimes, radioactive emissions, strength of materials, flood analysis, seismic analysis, and rainfall analysis.

Description: The Extreme Value Type A distribution is a continuous distribution that is unbounded. Its shape is always the same but it can be shifted or scaled to need. It is also called the Gumbel distribution.

Extreme Value Type B

Syntax: ExtremeValueTypeB(*mean*, *standard deviation*)

Applications: The Extreme Value distribution describes the limiting distribution of the extreme values of many types of samples. The Extreme Value Type B describes the limiting distribution for the *lowest* value samples whereas the Extreme Value Type A distribution describes the limiting distribution for the *highest* value samples. The Extreme Value Type B distribution is often used to represent parameters in growth models, astronomy, human lifetimes, radioactive emissions, strength of materials, flood analysis, seismic analysis, and rainfall analysis.

Description: The Extreme Value Type B distribution is a continuous distribution that is unbounded. Its shape is always the same but it can be shifted or scaled to need. It is also called the Gumbel distribution.

Gamma Distribution

Syntax: Gamma(*mean, standard deviation*)
Gamma(*shape*)

Applications: The Gamma distribution is often used to model nonnegative random variables, such as the time for a human operator to complete a task. The Gamma distribution is also used to represent lifetimes, lead times, personal income data, a population about a stable equilibrium, and service times. Use the Gamma distribution for task times when a task cannot be done much faster than the mean time, yet could take much longer.

Description: The Gamma distribution is a continuous distribution that is bounded at the lower side. It has three distinct regions. The range of values above the mean is much greater than the range of values below the mean.

Geometric Distribution

Syntax: Geometric(*mean*)

Applications: The Geometric distribution is used to represent the number of failures before the first success in a sequence of independent Bernoulli trials. The Geometric distribution can be used to represent the number of items inspected before encountering the first defective item, the number of items in a batch of random size, or the number of items demanded from an inventory.

Description: The Geometric distribution is a discrete distribution with a lower bound at 0 and no bound on the high side.

Inverse Gaussian Distribution

Syntax: InverseGaussian(*mean, standard deviation*)

Applications: The Inverse Gaussian distribution is used to model Brownian motion and diffusion processes with boundary conditions and to model the distribution of particle size in aggregates, reliability and lifetimes, and repair time.

Description: The Inverse Gaussian distribution is a continuous distribution that is bounded on the lower side. It is uniquely zero at the minimum x and always positively skewed. The inverse Gaussian is also called the Inverse Normal Distribution or Wald Distribution.

Logistic Distribution

Syntax: Logistic(*mean, standard deviation*)

Applications: The Logistic distribution is most often used as a growth model for populations, for weight gain, and for business failure. Occasionally, the Logistic function is used in place of the Normal function where exceptional cases play a larger role.

Description: The Logistic distribution is a continuous distribution that is unbounded and symmetrical about its mean. The shape of the Logistic distribution is very much like the Normal distribution, except that the Logistic distribution has broader tails.

LogLogistic Distribution

Syntax: LogLogistic(*scale, shape*)

Applications: The LogLogistic distribution is used to model the output of complex processes such as business failure and product cycle times.

Description: The LogLogistic is a continuous distribution that is bounded on the lower side. A true Loglogistic distribution occurs when the shape parameter is set to 1.

Lognormal Distribution

Syntax: Lognormal(*mean, standard deviation*)

Applications: The Lognormal distribution describes values with normally distributed natural logarithms. It is often used to describe random processes that represent the product (rather than the sum) of several small, independent events. The Lognormal distribution is used in many different areas, including the distribution of particle size in naturally occurring aggregates, dust concentration in industrial atmospheres, duration of sickness absence, lifetime distributions in reliability, distribution of income, employee retention, and many applications modeling weight, and height.

Use the Lognormal distribution for tasks that cannot be done much faster than the mean but sometimes take much longer. Because the Lognormal distribution generally has a longer tail than the Gamma distribution, it may be more appropriate for tasks with no known upper bound on their time duration in extreme situations.

Description: The Lognormal distribution is a continuous distribution that is bounded on the lower side. It is always 0 at minimum x, rising to a peak that depends on both the mean and standard deviation, and then decreases monotonically for increasing x.

Negative Binomial Distribution

Syntax: NegativeBinomial(*successes, probsuccess*)

Applications: The Negative Binomial distribution provides a good approximation for the sum or mixing of other discrete distributions. By itself, it is used to model accident statistics, birth-and-death processes, market research and consumer expenditure, lending library data, and biometrical data.

Description: The Negative Binomial distribution is a discrete distribution that is bounded on the low side at 0 and unbounded on the high side.

Normal Distribution

Syntax: Normal(*mean*, *standard deviation*)

Applications: The Normal distribution is commonly used for errors of various types, such as in the impact point of a bomb or quantities that are the sum of a large number of other quantities.

Description: The Normal distribution is a continuous distribution that is unbounded. It is also known as the bell curve. In a Normal distribution, each value is equally likely to fall on either side of the mean. A plot of the frequency for the occurrence of each value is bell-shaped, with the highest point of the bell occurring at the mean value.

The Normal distribution is frequently used to represent symmetrical data, but is unbound in both directions. If the data is known to have a lower bound, it may be better represented by suitable parameterization of the Lognormal, Weibull, or Gamma distributions. If the data is known to have both upper and lower bounds, the Beta distribution can be used.

Pareto Distribution

Syntax: Pareto(*minimum*, *scale*)

Applications: The Pareto distribution has historically been used to represent the income distribution of a society. It is also used to model many empirical phenomena with very long right tails, such as city population sizes, occurrence of natural resources, stock price fluctuations, size of firms, brightness of comets, and error clustering in communication circuits.

Description: The Pareto distribution is a continuous distribution that is bounded on the lower side. It has a finite value at the minimum x and decreases monotonically for increasing x.

Pearson Type 5 Distribution

Syntax: PearsonType5(*scale*, *shape*)

Applications: The Pearson Type 5 distribution is useful for modeling time delays where some minimum delay value is almost assured and the maximum time is unbounded and variably long. Examples include the time to complete a difficult task, time to respond to an emergency, and time to repair a tool, and so on.

Description: The Pearson Type 5 distribution is a continuous distribution that is bounded on the lower side. The Pearson Type 5 distribution is sometimes called the Inverse Gamma distribution due to the reciprocal relationship between a Pearson Type 5 random variable and a Gamma random variable.

The Pearson Type 5 distribution starts slowly near its minimum and has a peak slightly removed from it. With decreasing shape parameter, the peak gets flatter and the tail gets much broader.

Pearson Type 6 Distribution

Syntax: PearsonType6(*scale*, *shape1*, *shape2*)

Applications: The Pearson Type 6 distribution is rarely used, except in its reduced form as the F distribution. In this case, the distribution is used in the analysis of variance and is a function of the ratio of two independent random variables each of which has a chi-square distribution and is divided by its number of degrees of freedom.

Description: The Pearson Type 6 distribution is a continuous distribution that is bounded on the low side. It has three distinct regions dependent on the scale and shape factors.

Poisson Distribution

Syntax: Poisson(*mean number of occurrences per interval*)

Applications: The Poisson distribution models the number of independent events that occur in an interval of time. This includes many types of events in time or space, such as arrivals of telephone calls, defects in semiconductors manufacturing, defects in all aspects of quality control, molecular distributions, stellar distributions, geographical distributions of plants, shot noise, and so on. Note that the time between arrivals (defects) is Exponentially distributed, which makes this distribution a particularly convenient starting point even when the process is more complex.

Description: The Poisson distribution is a discrete distribution that is bounded at 0 on the low side and unbounded on the high side.

Rectangular (Uniform) Distribution

Syntax: Rectangular(*mean, minimum*)

Applications: The Rectangular distribution is used to represent a random variable with constant likelihood of being in any small interval between minimum and maximum and centered around the mean. The Rectangular distribution is primarily used to represent mechanical tasks rather than natural processes.

For example, when a pedestrian encounters a Don't Walk light at an intersection, the amount of time she has to wait for a Walk light is rectangularly distributed. The wait could take zero seconds or as long as the maximum length of the Don't Walk light.

Description: The Rectangular distribution is a continuous distribution bounded on both sides. It is a special case of the Beta distribution. It is also called the Uniform distribution. Most random number generators provide samples from the Uniform distribution on (0,1) and then convert these samples to random variates from other distributions.

Triangular Distribution

Syntax: Triangular*(mode, minimum, maximum*)

Applications: The Triangular distribution is often used when no or little data is available; it is rarely an accurate representation of a data set. However, it is employed as the function form of regions for fuzzy logic due to its ease of use.

Description: The Triangular distribution is a continuous distribution that is bounded on both sides. The Triangular distribution can take on very skewed forms including negative skewness. For the exceptional cases where the mode is either the minimum or maximum, the Triangular distribution becomes a right triangle.

Weibull Distribution

Syntax: Weibull(*scale, shape*)

Applications: The Weibull distribution is often used to model the strength of materials or the mean time to failure for a device. Industrial applications of survival analysis often involve testing components to destruction after subjecting them to a stress which is assumed to speed up the aging process. In particular, the Weibull distribution is used to represent wear-out lifetimes in reliability, duration of industrial stoppages, and thunderstorm data.

Description: The Weibull distribution is a continuous distribution that is bounded on the lower side.

■ Types of Probability Distributions

14

OptQuest

In today's highly competitive global environment, people are faced with many difficult decisions, such as allocating financial resources, building facilities, managing inventories, determining product mix strategies, and more. Modeling a decision problem in Micro Saint Sharp tells you what performance measures you can expect using a certain strategy.

A strategy can be defined as a certain set of values for the decision variables. Other words for strategy include scenario and solution. Note that solution refers to the decision variable values (inputs to the simulation), *not* the resulting values of the performance measures.

Suppose your goal was to find the best strategy (without using OptQuest). You would have to enter decision variable values for each strategy you were considering into Micro Saint Sharp and analyze the results. Then, you would have to repeat this process for every strategy you wanted to test. Often, it would not be clear how to adjust the decision variables from one simulation to the next. This type of search is tedious and, in problems with thousands or millions of potential alternatives, impractical.

OptQuest enhances Micro Saint Sharp by automatically searching for and finding optimal or near-optimal solutions to simulation models. Simulation models by themselves can only present a range of possible outcomes for any situation. They do not tell you how to control the situation to achieve the best outcome.

OptQuest, through a new optimization technique, finds the right combination of variables that produces the best results possible. If you use simulation models to answer questions such as, “How many machines of each kind should we purchase?” you can find the answer that maximizes performance (for example, throughput) while meeting budget and other constraints. If you asked, “How is performance of an emergency room effected by changes of the staff levels?” you can additionally determine how to schedule doctors and nurses to maximize performance. And if you wonder, “Which routing rules should I select to send data over a network?” you can select the one that utilizes the network more effectively, for example, by minimizing operating costs.

How OptQuest Interacts with Micro Saint Sharp

OptQuest works with Micro Saint Sharp by sharing variables between Micro Saint Sharp and OptQuest. You can directly choose the variables that will have an effect on the Micro Saint Sharp results. This includes every aspect of an optimization model including constraints, decision variables, and the optimization problem itself.

When the optimization runs:

1. OptQuest feeds a potential solution to the Micro Saint Sharp model by setting the decision variable values.

2. Micro Saint Sharp runs one simulation to evaluate the solution.

3. OptQuest takes the resulting response values from Micro Saint Sharp.

4. OptQuest analyzes the results of the simulation and uses its intelligent search procedures to generate a new potential solution, which it then sends to Micro Saint Sharp.

OptQuest repeats this process. The ultimate goal is to find the solution that optimizes (maximizes or minimizes) the value of the model's objective.

Elements of an OptQuest Optimization Model

An optimization model in OptQuest for Micro Saint Sharp has three major elements:

- Decision variables
  Decision variables are quantities over which you have control, such as the amount of product to make, the number of dollars to allocate among different investments, or which operational rules to select from a limited set.

- Constraints
  Constraints describe relationships among decision variables that restrict the values of the decision variables. For example, a constraint might ensure that the total amount of money allocated among various investments cannot exceed a specified amount, or might ensure that no more than one operational rule from a certain group can be selected.

- Objective
  The objective presents a mathematical representation of the optimization model's objective, such as maximizing profit or minimizing cost, in terms of the decision variables.

Overview of Using OptQuest

To use OptQuest:

1. Create a simulation model with Micro Saint Sharp.

2. Prepare the model for optimization.

   For details, see "Preparing the Model" on page 318.

3. Set up the optimization.
   - Select model variables to use in the optimization.
   - Define constraints.
   - Define the objective.
   - Select optimization options.

   For details, see “Setting up the Optimization” on page 319.

4. Run the optimization.

   For details, see “Running the Optimization” on page 329.

5. Interpret the results

   For details, see “Interpreting the Results” on page 330.

Preparing the Model

Decision variables are variables that you can control, for example, the number of machines of each type in a job shop. In Micro Saint Sharp, all aspects of an OptQuest optimization are defined from the Tree View window.

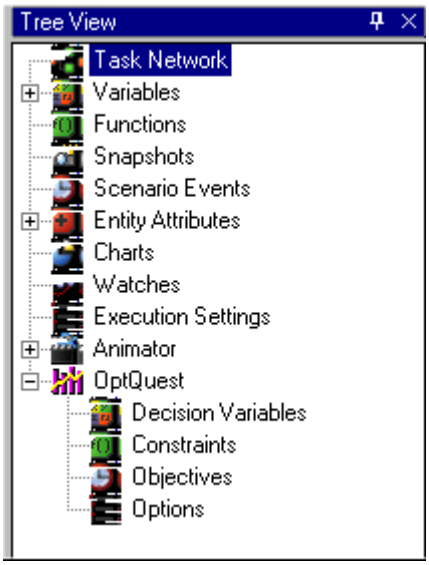
The values of the objective function and the values of output constraints depend on the output of the simulation. Therefore, they are based on responses.

You define an objective function in Micro Saint Sharp by creating a mathematical formula involving multiple response statistics. This objective formula should return a variable whose goal is either to be minimized or maximized.

Setting up the Optimization

The components of an optimization automatically display in the Tree View in Micro Saint Sharp. You create an optimization by opening your Micro Saint Sharp model and entering the decision variables, constraints, objectives, and options for the optimization.

OptQuest components include Decision Variables, Constraints, Objectives, and Options. These components are discussed in the following sections.



Decision Variables

Decision variables are the variables or resources in your model that you control. After you define the decision variables in your simulation model, you can select which decision variables to optimize in OptQuest. The values of these decision variables will change with each simulation until OptQuest finds values that yield the best objective. For some analyses, you might fix the values of certain decision variables and optimize the rest.

Displaying Decision Variables

To display defined decision variables:

In the tree view, click the Decision Variables item to expand it.

A list of defined variables displays in the tree view.

Adding Decision Variables

To add a decision variable:

1. In the tree view, right-click the Decision Variables item and select **Add Decision Variable**.

   A new decision variable is added to the tree view.

2. Click the new variable.

   The Properties window for the decision variable displays.

Options include the following:

- Initial value. Contains the initial value OptQuest uses to start optimizing.
- Lower Bound. Contains the lower limit for the decision variable.
- Name. Displays the name of the decision variable from your model. You can select the variable name from the drop-down list box.
- Notes. Displays any notes pertaining to this decision variable.
- Step Size. For integers, contains the value used to assign to the decision variable. The step size allows you to modify the feasible values in a range for an integer variable. For example, a discrete variable defined in the range -5 to 11 has the following feasible values when the step size is equal to 1:

  -5 -4 -3 -2 -1 0 1 2 3 . . .11

  However, if you change the step size to 5, the feasible values for the variable are reduced to the following:

  -5 0 5 10
- Type. Indicates whether the variable is continuous (real) or integer. You can select either Continuous or Integer from the drop-down menu. You must specify a step size for integer decision variables.
- Upper Bound. Contains the upper limit for the decision variable.

Editing Decision Variables

To edit a decision variable:

1. Click the Decision Variable item in the tree view.
2. In the tree view, click the decision variable to edit.
3. Edit the decision variable properties in the Properties window.

Deleting Decision Variables

To delete a decision variable:

1. In the tree view, click the Decision Variables node to expand it and then click the decision variable to delete.

2. Press the Del key on the keyboard. You can alternatively right-click the variable and select **Delete Decision Variable** from the menu that displays.

Constraints

Constraints are relationships among decision variables and output variables. For additional information on constraints, see “Using Constraints” on page 324.

Displaying Constraints

To display defined constraints:

In the tree view, click the Constraints item to expand it.

A list of defined constraints displays in the tree view.

Adding Constraints

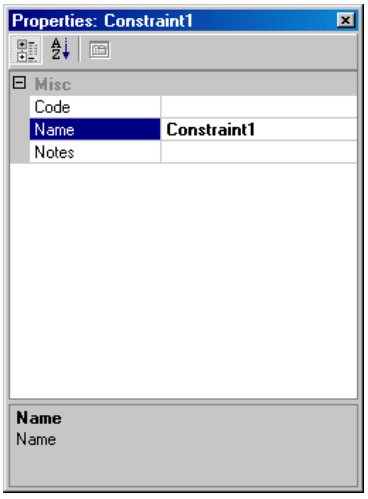
To add a constraint:

1. In the tree view, right-click the Constraints item and select **Add Constraint** from the menu that displays.

   A new constraint is added to the tree view.

2. In the tree view, click the new constraint.

   The Properties window for the constraint displays. Options include Code, Name, and Notes.



3. Click the Code field and enter the constraint definition. Constraints should be in standard mathematical formats.

Editing Constraints

To edit a constraint:

1. Click the Constraints item in the tree view.
2. In the tree view, click the constraint to edit.
3. Edit the constraint in the Properties window.

Deleting Constraints

To delete a decision variable:

1. In the tree view, click the Constraints node to expand it and then click the constraint to delete.

2. Press the Del key on the keyboard. You can alternatively right-click the variable and select **Delete Constraint** from the menu that displays.

Using Constraints

You use constraints to restrict the values of decision variables by defining relationships among them. For example, if the total budget for scanners ($2000 each) and printers ($1000 each) is $5000, you can define this as:

```
2000*scanners + 1000*printers <= 5000
```

In another example, if `x1` represents the length of the queue in task 1 and `x2` represents the length of the queue in task 2, the following constraint restricts the search to solutions for which queue two's length is at least as long as task one's queue.

```
x2 - x1 >= 0
```

Constraints can also be defined to restrict the values of output variables. Constraints on output variables are sometimes called probabilistic constraints, chance constraints, or goals in other literature.

Constraint Feasibility

A feasible solution is one that satisfies all constraints. Infeasibility occurs when no combination of values of the decision variables can satisfy a set of constraints. If a constraint is defined using only decision variables, and the constraint is infeasible, OptQuest will tell you there is an infeasible constraint and will not start the optimization.

If a constraint involves output variables it is called an output constraint. When a constraint involves output variables, a simulation must be run in order to determine feasibility of the constraint. OptQuest cannot determine if every solution will produce an infeasible result. If the constraint is infeasible using a solution suggested by OptQuest, the result is marked as infeasible.

✓ Note:
Constraints defined using only decision variables can narrow the search for the best solution thus improving OptQuest's performance.

Objectives

Use the objective to define the objective of your optimization. The objective is an expression that represent the model's objective, such as minimizing queues or maximizing profits.

Displaying Objectives

To display defined objectives:

In the tree view, click the Objectives item to expand it.

A list of defined objectives displays in the tree view.

Adding Objectives

You can use as many objective functions as you want.

To add an objective:

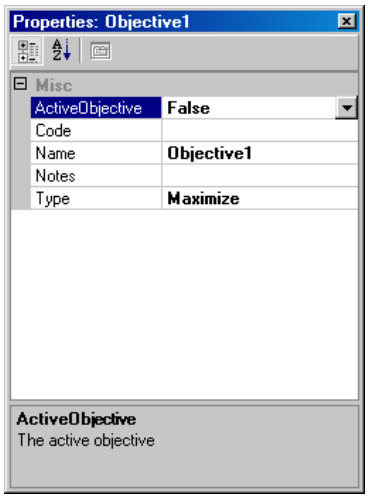
1. Right-click the Objectives item in the tree view and select **Add Objective** from the menu that displays.

   A new objective is added to the tree view.

2. Click the new objective.

   The Properties window for the objective displays. Options include the following:

   - ActiveObjective. Indicates whether the objective is to be used. Select True for the objective function that you want OptQuest to consider when running. You can have multiple objective functions, but only one can be active at a time.
   - Code. Contains the objective function.
   - Name. Contains the name of the objective.
   - Notes. Contains any notes pertaining to the objective.
   - Type. Indicates whether the intent is to maximize or minimize the objective.



3. Enter or select the options for the objective.

Editing Objectives

To edit an objective:

1. Click the Objectives item in the tree view.
2. In the tree view, click the objective to edit.
3. Edit the objective in the Properties window.

Deleting Objectives

To delete an objective:

1. In the tree view, click the Objectives node to expand it and then click the objective to delete.
2. Press the Del key on the keyboard. You can alternatively right-click the variable and select **Delete Objective** from the menu that displays.

Options

Use the Options item to set options to control the optimization process.

To select optimization options:

1. Click the Options item in the tree view.

   The Properties window for the options displays. Options include the following:

   - Iterations. Displays the number of times to run the optimization.
   - Name. Displays the name of the Options setting.
   - Notes. Contains any notes pertaining to the option setting.
   - Run Time. Contains the number of minutes to run the optimization.

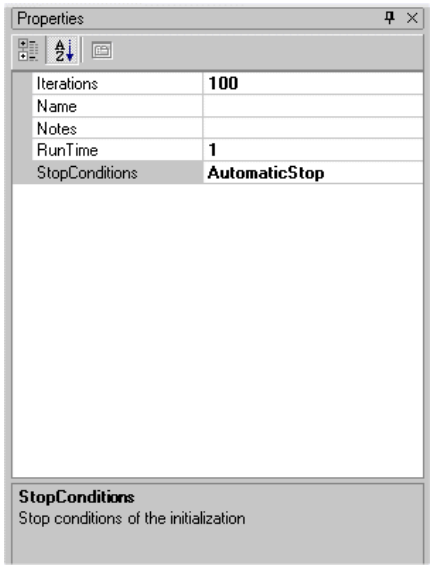
- Stop Conditions. Indicates how the optimization will be stopped.

  Manual Stop. When you select this option, you manually stop the optimization using the Halt Optimization command.

  Automatic Stop. When you select this option, OptQuest assesses the opportunities for additional improvement of the objective value and based on this assessment determines whether it is time to stop.

  Iteration Count. When you select this option, the number of simulations that you enter in the Iterations field is run. The default is 100 simulations.

  Maximum Time. When you select this option, the simulation is run for the amount of minutes you enter in the Run Time box.



2. Enter or select the options for the optimization.

Running the Optimization

To start an optimization:

From the **Optimization** menu, select **Start Optimization**.

To halt an optimization:

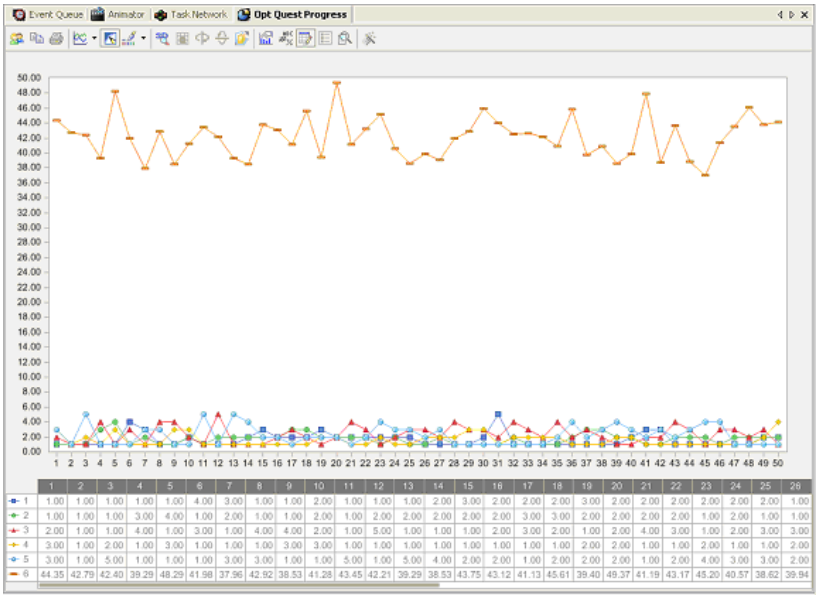
From the **Optimization** menu, select **Halt Optimization**.

To check optimization for errors:

From the **Execution** menu, select **Check for Errors**.

Performance Graph

When you run an optimization, the OptQuest Progress window automatically displays. This window displays a performance graph, a plot of the objective values and the decision variable values as a function of the run number. As OptQuest runs, this window graphically displays values as they are tested. Upon completion, you can scroll through all of the decision variable values that were tested.



Interpreting the Results

The Micro Saint Output window displays the best optimization value and the values of the decision variables from the run. Decision variables that take on the same or almost always the same value are considered determined variables. If the best solution's decision variables are determined variables, the solution is likely robust.

You will have to determine the robustness of the results and decide whether to run the optimization for a longer period of time or for more simulation runs. Using the optimal values of the decision variables, you can more accurately assess the risks of the recommended solution.

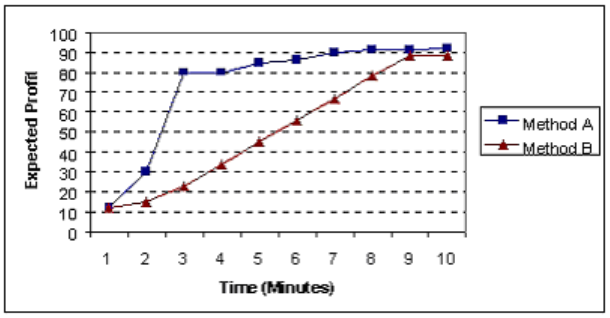
To more accurately assess the recommended solution, run a longer Micro Saint Sharp simulation using the optimal values of the decision variables. Depending on the simulation model, it may be more meaningful to perform multiple runs of the simulation instead of running one long simulation. For additional information on improving the optimization performance, please see the following section titled Suggestions for Using OptQuest.

Suggestions for Using OptQuest

There are many factors that influence the performance of OptQuest. For example, consider two optimization methods, A and B, applied to a problem with the objective of maximizing expected returns. When you evaluate the performance of each method, you must consider which method satisfies the following criteria:

- Finds a solution with a larger expected return
- Jumps to the range of high-quality solutions faster

Below is the Performance Graph for the two hypothetical methods.



This illustration shows that, although both methods find solutions with a similar expected profit after 10 minutes of searching, method A jumps to the range of high-quality solutions faster than B. For the criteria listed above, method A performs better than method B.

While using OptQuest, you will obtain performance profiles similar to method A. OptQuest's search methodology is very aggressive and attempts to immediately find high-quality solutions, causing large improvements, (with respect to the initial solution), early in the search. This is critical when OptQuest can perform only a limited number of simulations.

However, several factors affect OptQuest's performance, and the importance of these factors varies from one situation to another. This section reviews these factors and offers tips and suggestions on how to achieve maximum performance.

Factors that Affect Search Performance

Any heuristic method for solving problems cannot guarantee that it will find the optimal solution. It might only find a solution that is close to the optimal solution, usually referred to as the best solution; this is why maximizing performance is critical.

The following is a list of relevant factors that directly affect search performance. These factors are discussed in the following sections.

- Number Of Decision Variables
- Initial Values
- Bounds and Constraints
- Complexity of the Objective
- Constraints
- Feasibility
- Number of Replications and Simulations

- Simulation Accuracy
- Simulation Speed

Number Of Decision Variables

The number of decision variables greatly affects OptQuest's performance. OptQuest has no physical limit on the number of decision variables you can use in any given problem. However, the performance might deteriorate if you use more than 100 decision variables.

Also, as the number of decision variables increases, you need more simulations to find high-quality solutions. General guidelines for the minimum number of simulations required for a given number of decision variables in a problem are the following:

- For less than 10 decision variables, the minimum number of simulations is 100
- Between 10 and 20 decision variables, the minimum number of simulations is 500
- Between 20 and 50 decision variables, the minimum number of simulations is 2,000
- Between 50 and 100 decision variables, the minimum number of simulations is 5,000

For very large numbers of decision variables, you might try this procedure:

- Decrease the number of replications per simulation, at least initially
- Run the optimization to get an approximate solution
- Set the suggested values to the approximate solution
- Further restrict the bounds on the decision variables

- Increase the number of replications to increase accuracy
- Rerun the optimization

You might de-select certain decision variables and optimize the rest. If you have already run an optimization, you may have information about which decision variables have the least effect on the objective function. When you de-select one or more decision variables and rerun the optimization, the search focuses on the remaining, more important, decision variables.

Initial Values

The initial values are the values listed as the Suggested Values of the decision variables interface. Initial values are important because the closer they are to the optimal value, the faster OptQuest can find the optimal solution. If the initial values are constraint-infeasible, they are ignored.

For potentially large models with many decision variables, you might find it helpful to first run a simplified version of the optimization to find initial values for the full-blown model. For example, you might use expected values for some of the random variables in the model.

Bounds and Constraints

You can significantly improve OptQuest's performance by selecting meaningful bounds for the decision variables. Suppose, for example, that the bounds for three decision variables (X, Y, and Z) are:

```
0 <= X <= 100
0 <= Y <= 100
0 <= Z <= 100
```

And in addition to the bounds, there is the following constraint:

```
10*X + 12*Y + 20*Z <= 200
```

Although the optimization model is correct, the decision variables bounds are not meaningful. A better set of bounds for these decision variables would be:

```
0 <= X <= 20
0 <= Y <= 16.667
0 <= Z <= 10
```

These bounds take into consideration the values of the coefficients and the constraint limit to determine the maximum value for each decision variable. The new "tighter" bounds result in a more efficient search for the optimal values of the decision variables. However, this efficiency comes at the expense of missing the optimal solution if it lies outside the specified bounds.

Complexity of the Objective

A complex objective has a highly nonlinear surface with many local minimum and maximum points.

OptQuest is designed to find global solutions for all types of objectives, especially complex objectives. However, for more complex objectives, you generally need to run more simulations to find high-quality global solutions.

Constraints

Constraints can be used to restrict the values of decision variables by defining relationships among the decision variables; constraints can also restrict the value of output variables.

If a constraint is defined using only decision variables, OptQuest can eliminate sets of decision variables values that are constraint-infeasible before it runs the simulation. Limiting the optimization by defining constraints on decision variables is extremely time-effective.

If a constraint contains an output variable, a simulation must be run to determine whether the suggested solution satisfies the constraint.

The search process benefits from the use of constraints on decision variables and tight bounds on decision variables. However, performance generally suffers when you include output constraints in the optimization model for two reasons:

- Output constraints are very time-consuming to evaluate, since OptQuest must run an entire simulation before determining whether the results are constraint-infeasible.
- To avoid running constraint-infeasible simulations, OptQuest must identify the characteristics of solutions likely to be constraint-feasible; this makes the search more complex and requires more time.

Even though output constraints can greatly decrease the number of feasible simulations performed during an optimization, output constraints can focus your search to effectively rule out undesirable solutions.

When you use output constraints, you should increase the search time by at least 50% (based on the time used for an equivalent problem without output constraints).

If you have lots of output constraints that OptQuest can't easily satisfy, consider combining your output constraints into one multi-objective function.

Feasibility

A feasible solution is one that satisfies all constraints. Each time OptQuest generates a new set of values for decision variables, it tests the solution for constraint feasibility. Only constraint-feasible solutions are sent to Micro Saint Sharp for evaluation. OptQuest then uses the responses from Micro Saint Sharp to determine if an evaluated solution is constraint-feasible.

OptQuest makes finding a feasible solution its highest priority. Once it finds a feasible solution, it then concentrates on finding better solutions.

The fact that a particular solution may be infeasible does not imply that the problem itself is infeasible. However, infeasible problems do exist. For example, suppose that in a Job Shop problem a foreman insists on finding an optimal configuration with the following constraints:

```
drills + grinders <= 4
drills + grinders >= 5
```

Clearly, there is no combination that will satisfy both of these constraints.

Or, for this same example, suppose the bounds for a decision variable were:

```
3 <= saws <= 5
```

And a constraint was:

```
saws <= 2
```

This also results in an infeasible problem.

You can make infeasible problems feasible by fixing the inconsistencies of the relationships modeled by the constraints. OptQuest detects optimization models that are constraint-infeasible and reports them to you.

If a model is constraint-feasible, OptQuest will always find a feasible solution and search for the optimal solution (i.e., the best solution that satisfies all constraints).

Number of Replications and Simulations

When OptQuest runs an optimization, it runs a simulation to evaluate each set of decision variable values. The quality of the optimization results therefore depends on the number of simulation and the number of replications per simulations.

For a set period of time, the number of replications per simulation is inversely related to the number of simulations; as you increase one, the other decreases. Decreasing the number of replications can help increase the number of simulations.

The more simulations OptQuest can run, the more sets of values it can evaluate, and the more likely it is to find a solution close to the optimal solution.

Simulation Accuracy

Two factors affect simulation accuracy:

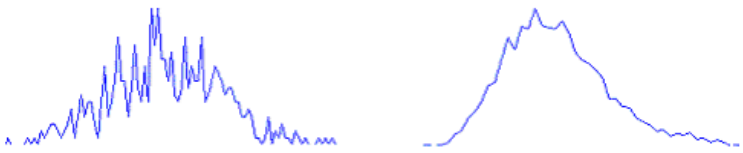
- Number of Replications per Simulation
- Objective Noisiness

Number of Replications per Simulation

For sufficient accuracy, you must set the number of replications per simulation to the minimum number necessary to obtain a reliable estimate of the objective function being optimized; this minimum number is typically found with empirical testing.

Objective Noisiness

Noisiness can also affect the accuracy of your OptQuest results.



Noisy objective **Smooth objective**

The objective on the left has significant amounts of noise caused by the probability distributions used to model the problem's uncertainty. For these types of objectives, OptQuest might have trouble discerning the minimum or maximum value. You can detect noisy functions by watching the solutions for best solutions that seem to "bounce around" from one set of values to completely different sets of values. To help solve this problem, you can increase the number of replications per simulation.

On the right, the objective appears smooth due to the relative certainty in the model assumptions. In these cases, OptQuest should quickly converge to the best solution.

Simulation Speed

Some suggestions for increasing speed include:

- Reduce the size of your model (or the time horizon of the simulation)
- Increase your system's RAM memory
- Reduce the number of uncertain elements in the simulation
- Close other applications

OptQuest Methodology

OptQuest implements certain procedures that make it possible to optimize simulations and complex systems. Although details of the actual implementation are proprietary information of OptTek Systems, Inc., this section provides an overview of the methods used by OptQuest.

Classical Methods

Optimization attempts to find the best solution to problems described with an objective function (to optimize) and a set of constraints (to restrict the values of the decision variables). The best-known optimization tool is linear programming, which requires that you express the objective function and constraints using linear functions. The techniques to solve linear programming problems are well developed, and robust commercial packages are readily available. Linear programming uses techniques to find optimal solutions to problems without evaluating all possible alternatives. Models with thousands and even millions of decision variables can be solved with reasonable amounts of computer time.

Unfortunately, not all business and industrial problems can be described with a linear objective and linear equalities or inequalities. Many complex systems might not even have a convenient mathematical representation, linear or nonlinear. To use techniques such as linear programming or its cousins (nonlinear programming and integer programming), you must simplify assumptions about the real system in order to properly frame the problem. One typical simplifying assumption disregards the "statistical fluctuations" of the system. For example, you might model an investment problem by assuming that returns on investment options do not vary, using a moment in time estimate for modeling purposes.

Once the problem is formulated, well-established techniques can often find an optimal solution, provided the formulation is "congenial." Still, there are situations where even the deterministic versions remain hard to solve because of the combinatorial nature of these problems; production scheduling is one example. Suppose you have the goal of determining an optimal order to process a set of jobs. Even with fewer than a hundred jobs to order, the number of alternative configurations is astronomical. (Using all of today's computers working at once, it would take longer than the age of the universe to enumerate seventy jobs!)

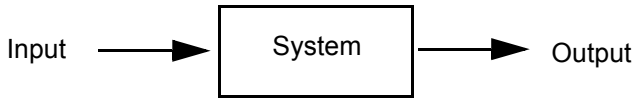
Metaheuristics

Since practical problems are not becoming easier to solve, and practitioners need solutions to increasingly complex problems, researchers have developed solution procedures called metaheuristics, which guide other heuristics to search for solutions. Heuristics provide approximate solutions to complex problems. For example, a prod**u**ction heuristic might give priority to jobs with the shortest estimated processing time. Depending on the context, this heuristic (or processing rule) might actually work fairly well. However, in some other situations the results might be disastrous (with dire consequences for equipment utilization, production lead times, and work-in-process inventory).

The alternatives were less than encouraging: either seek optimal solutions to simplified problems or seek sub-optimal and possibly very poor solutions to complex systems. Metaheuristics arose with the goal of providing something better. The aspiration was to integrate intelligent procedures and fast computer implementations to address complex problems (without simplifying them), and to provide high-quality solutions. Dedicated study and a number of key innovations achieved useful advances.

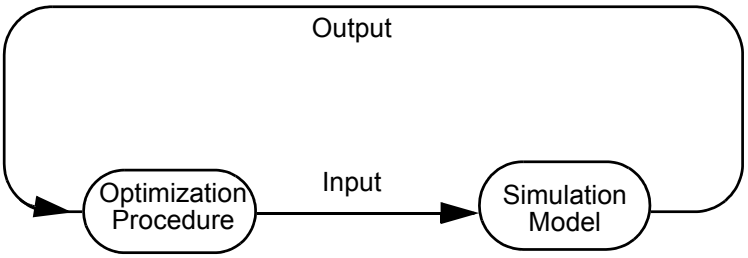
However, the successful applications typically required highly problem-specific designs, customizing the solution procedures to each particular case. Then, every time a new problem surfaced, a new procedure had to be developed. The metaheuristic approaches are based on general principles, but they also owe their efficiency to the knowledge of characteristics particular to each situation. In this sense, there is no separation between the model and the solution procedure. In fact, the solution procedure might be seen as a way of modeling the problem.

If it can be done successfully, it is better to separate the solution procedure from the system to optimize. The disadvantage of this black box approach is that the optimization procedure is generic and does not know anything about what goes on inside the box.



How OptQuest Uses Metaheuristics

The clear advantage to the OptQuest approach is that you can use the same optimizer for many systems. OptQuest is a generic optimizer that successfully separates the method from the model. In this case, the optimization problem is defined outside the system (represented by a simulation model). This design lets you create a model of your system that includes as many elements as necessary to accurately represent the real thing. While the simulation model can change and evolve to incorporate additional elements, the optimization routines remain the same. Hence, there is a complete separation of the model that represents the system, and the procedure that solves optimization problems defined within this model.



The optimization procedure uses the outputs from the simulation model to evaluate the inputs to the model. Analyzing this evaluation and previous evaluations, the optimization procedure selects a new set of input values. The optimization procedure performs a special non-monotonic search, where the successively generated inputs produce varying evaluations, not all of them improving, but which over time provide a highly efficient trajectory to the best solutions. The process continues until it reaches some termination criterion (usually a time limit).

Scatter Search and Tabu Search

Two of the best-known metaheuristics are genetic algorithms and Tabu search. Genetic Algorithm (GA) procedures were developed by John Holland in the early 1970s at the University of Michigan. Parallel to the development of GAs, Fred Glover of OptTek Systems, Inc. established the principles and operational rules for Tabu search (TS) and a related methodology know as Scatter search.

Scatter search, which was introduced in roughly the same period as the early GA proposals, has some interesting commonalities with GA ideas, although it also has a number of very distinct features. Several of these features have been incorporated into GA approaches, after an intervening period of approximately a decade, while others remain largely unexplored in the GA context.

Scatter search operates on a set of points, called *reference points,* that result in good solutions. The approach systematically generates linear combinations of the reference points to create new points, each of which maps into an associated point that yields integer values for discrete variables. *Tabu search* is then superimposed to control the composition of reference points at each stage.

Tabu search has its roots in the field of Artificial Intelligence. Memory is a fundamental concept in Tabu search, which uses search history to guide the process. In its simplest form, memory prohibits the search from reinvestigating solutions that have already been evaluated. However, the use of memory in OptQuest is much more complex and uses memory functions to encourage search diversification and intensification. These memory components let the search escape from locally optimal solutions to find a globally optimal solution.

Similarities are immediately evident between Scatter search and the original GA proposals. Both are *population-based* approaches, and both produce the new elements from some combination of existing elements. On the other hand, several contrasts between these methods may also be noted. The early GA approaches chose parents randomly to produce offspring, and introduced randomization to determine which components of the parents to combine.

By contrast, Scatter search does not revert to randomization by being indifferent to choices among alternatives. However, as in probabilistic Tabu search, the approach incorporates strategic probabilistic biases, taking account of evaluations and history. Scatter search focuses on generating relevant outcomes while still producing diverse solutions, due to the way the generation process (rounded linear combinations) is implemented. In particular, Scatter search considers that the generation of new points might contain information that is not contained in the original points.

To learn more about Scatter search, see "Scatter Search and Star-Paths: Beyond the Genetic Metaphor," Fred Glover, *OR Spektrum*, 1995, vol. 17, pp. 125-137.

To learn more about Tabu search, see "Tabu Search", Fred Glover and Manuel Laguna, Kluwer Academic Publishers.

How OptQuest Uses Scatter Search

Scatter search is an information-driven approach, exploiting knowledge derived from the search space, high-quality solutions found within the space, and trajectories through the space over time. The combination of these factors creates a highly effective solution process. The incorporation of such designs gives OptQuest the ability to solve complex simulation-based problems with unprecedented efficiency.

OptQuest Glossary

Best Solution

The set of decision variable values that achieves the best outcome found by a heuristic search. Heuristic methods cannot guarantee the best solution to be optimal.

Bound

A maximum or minimum limit you set for each decision variable.

Constraint

A limitation that restricts the decision variables and output variables by defining relationships among them. For example, if the total budget for scanners ($20,000 each) and printers ($10,000 each) is $50,000, you can define this as:

```
20000*scanners + 10000*printers <= 50000
```

In another example, if x1 represents the number of spaces in buffer 1 and x2 represents the number of spaces in buffer 2, the following constraint restricts the search to solutions for which buffer 2 has at least as many spaces as buffer 1.

```
x2 - x1 >= 0
```

Constraints on decision variables can narrow the search for the optimal solution, thus improving OptQuest's performance. Solutions that violate a constraint on decision variables are not sent to your simulation for evaluation.

Output constraints contain one or more output variables and must be evaluated after a simulation has been run.

Constraint Feasible

A feasible solution is one that satisfies all constraints. Infeasibility occurs when no combination of values of the decision variables can satisfy a set of constraints. If a constraint is defined using only decision variables, and the constraint is infeasible, OptQuest will tell you there is an infeasible constraint and will not start the optimization. If a constraint contains an output variable, OptQuest must run a simulation to determine feasibility. If the solution is infeasible, it is marked as infeasible in the status and solutions window.

Continuous Variable

A variable that can be fractional, so that any given range contains an infinite number of possible values.

Optimization models can be classified as:

- Discrete. Discrete decision variables only.
- Continuous. Continuous decision variables only.
- Mixed. Both discrete and continuous decision variables.

Design Variable

Design variables are used in problems to represent decisions where there is no clear sense of direction. For example, design variables can be used to represent an ID for an item such as a battery from a list of batteries. They are used when a larger number may not imply a better choice. For example, Battery #8 may not be a better choice than Battery #2. *See also* "Decision Variable" on page 347.

Discrete Variable

A variable that can only assume values equal to its lower bound, plus a multiple of its step size; the step size is any number greater than zero but less than the variable's range.

Optimization models can be classified as:

- Discrete. Only discrete decision variables.
- Continuous. Only continuous decision variables.
- Mixed. Both discrete and continuous decision variables.

Decision Variable

A variable or resource in your model that you control. After you define the decision variables in your simulation model, you can select which decision variables to optimize in OptQuest. The values of these decision variables will change with each simulation until OptQuest finds values that yield the best objective. For some analyses, you might fix the values of certain decision variables and optimize the rest.

Heuristic

An approximate and self-educating technique for improving solutions to complex problems. A heuristic may be a rule of thumb, simplification, or educated guess that reduces or limits the search for solutions to problems. Heuristics do not guarantee optimal, or even feasible, solutions.

Linear

A formula or equation in which all variables are only multiplied by constants, not by other variables (including themselves). For example, `3x - 1.2y` is a linear relationship since both the first and second term only involve a constant multiplied by a variable.

OptQuest can handle linear or nonlinear objectives.

Metaheuristic

A family of optimization approaches that includes Scatter search, genetic algorithms, simulated annealing, Tabu search, and their hybrids.

Multiobjective Optimization

A technique that combines multiple, often conflicting objectives, such as maximizing returns and minimizing risks, into one objective.

Nonlinear

Terms such as x^2, `xy`, or `1/x` make nonlinear relationships. An optimization model can be linear or nonlinear, depending on the form of the objective and constraints. Any models that contain nonlinear terms are classified as nonlinear. OptQuest can handle linear or nonlinear objectives.

Objective

A formula in terms of decision variables and output variables that provides a mathematical representation of the model's objective function.

Optimization

A process that searches for the optimal solution to a model.

Optimization Model

A model that seeks to maximize or minimize some quantity, such as service levels or cost.

Output Constraint

A constraint that contains one or more output variables. Since output constraints contain output variables, they must be evaluated after a simulation has completed and the values of the outputs can be retrieved. Input constraints contain only decision variables and can be evaluated by OptQuest before running a simulation.

Performance

For an optimization program, the ability to find high-quality solutions as fast as possible.

Probability Distribution

A set of all possible events and their associated probabilities.

Replication

A single run of the model, starting with a single set of decision variables values and producing a single value for the objective function response. If the model contains uncertain elements, then an evaluation (simulation) of a single set of decision variables values should involve multiple replications to get a more reliable estimate of the objective function value.

Response

An output from the model.

Scatter Search

A population-based metaheuristic that operates on a collection of reference points, which results in good solutions to a problem. The Scatter search approach systematically generates linear combinations of the reference points to create new points, each of which maps to an associated point that yields integer values for discrete variables.

Sequence Variable

Sequence variables are used to solve sequencing problems such as the order in which paint should be mixed to minimize cleanup time between color changes. The value of the sequence variable represents the location within the sequence.

Simulation (or experiment)

A set of replications using the same set of decision variables values, intended to evaluate the expected value of the objective function. OptQuest finds the best objective by running many experiments or simulations using different sets of decision variable values.

Solution

A certain set of values, such as a strategy or scenario, for the decision variables. Note that *solution* refers to the decision variable values, not to the resulting values of the performance measures.

Tabu search

A metaheuristic that uses search history and memory management to guide the problem-solving process. In its simplest form, memory prohibits the search from reinvestigating solutions that have already been evaluated. In OptQuest, memory functions to encourage search diversification and intensification. These memory components divert the search from locally optimal solutions to find a globally optimal solution.

■ OptQuest Glossary

Glossary

Animator

A Micro Saint Sharp companion product that develops image animation for models. You can import a background, select from a large set of images, and build Animator instructions into your model. Then, you can watch the images move as the animation runs. Animator is an optional add-on component.

Array

An ordered set of variable values that are indexed to a single variable name. An array can be a one-dimensional list, a two-dimensional table of rows and columns, or a multi-dimensional array.

Clock

System variable that records elapsed time in simulation time units since the beginning of model execution. The `Clock` variable can be used in any expression in a model.

Decision Node

A diamond-shaped object on a network diagram containing one of the following letters: P, M, or T—representing a Probabilistic, Multiple, or Tactical decision.

Effect

An expression that executes as a result of task or queue activity during model execution. Tasks can have beginning, launch, and ending effects. Queues can have entering and departing effects. You can use these effects to change variable values and thus represent the system changes that occur as a result of the task or queue activity.

Entity

A conceptual object that travels through a task network and indicates by its location when each task or queue is executing or waiting to execute. The entity may represent a physical object, such as a part being built on a production line. It may also represent a person performing the tasks. Or, the entity may simply trace the sequence of tasks and not correspond to anything physical. During model execution, entities can be represented by symbols or numbers that travel through the network diagram. Entities are identified by corresponding values of its entity attributes.

Entity.Duration

Entity attribute that records the time each entity spends in the current task or queue.

Entity.Tag

Entity attribute that records the identity of each entity when there are multiple entities traveling through a network. Once an entity has a `Entity.Tag` value, the value stays with the entity through the remainder of model execution.

Event

Something that is scheduled to happen during a simulation at a specific clock time. Some events are expressions that you schedule to occur at specific times—these are called scenario events. All other events are scheduled, such as the finish times for currently executing tasks, as the model is running. You can watch events being scheduled by displaying the Event Queue window.

Event Queue

Events that are scheduled to happen during model execution are placed in an Event Queue. As each event occurs, it is removed from the top of the queue. These events are displayed in the Event Queue window.

Expression

A calculation, formula, function, or statement that supplies a value or performs an operation. Expressions can contain combinations of numerical values (constants or variables), functions, mathematical and logical operators, and logical statements.

Function

One or more mathematical or logical expressions that are assigned a single name and return a single value. When a function name is encountered in an expression in a model, the model executes the expressions included in the function and returns a value. Micro Saint Sharp contains model, mathematical, distribution, and Animator functions for your use or you can define and save your own functions with the model.

Job

Either a single task or a subnetwork. A network is made up of jobs linked by paths and sometimes preceded by queues.

Launch Effect

An expression that represents an effect of the task starting to execute. It is similar to a beginning effect except that it is evaluated after the task's execution time has been calculated. The launch effect was created to provide a place for Animator function calls, which often use entity attribute `Entity.Duration` as their time parameter. Because it stores the execution time for the current task, `Entity.Duration` is not valid in a beginning effect.

Network

A sequential relationship of jobs (tasks and subnetworks) that simulate a system, activity, or process. You create the network by drawing a network diagram, and you use variables to simulate the effects of the tasks on other tasks and on the system.

Network Diagram

A graphical depiction of a model network showing the sequence of tasks and subnetworks, queues, and the possible paths through the network. You draw the network diagram using the tools in the network diagram palette.

Path

A sequential connection between two jobs (tasks or subnetworks), so that one follows another under certain specified conditions. You draw the paths with the Path tool, then specify the conditions or probabilities for taking each path when you define the routing from each task to its possible following jobs.

Queue

A waiting area in front of a task or subnetwork where entities accumulate while they are waiting to execute the job. Each time the release condition for the job becomes true, an entity can be released from the queue into the job. Entities are released in the order you specify when you define the queue: First In, First Out (FIFO); Last In, First Out (LIFO); or sorted on a priority expression you specify.

Queue Priority

An expression to use in sorting the entities in a queue. When it is time to release an entity from the queue, the value of the Priority Expression for each entity in the queue is calculated. The entity with the highest priority value is released into the next job. For the priority expression to be meaningful, it must have different values for different entities. For this reason, the Priority Expression is usually based on the value of an entity attribute value for the entity in question.

Random Number Seed

A number between approximately -2 billion and +2 billion used to generate random numbers for calculating task execution times and taking probabilistic paths through the network. You supply the random number seed when you define the Execution settings for a model.

Release Condition

An expression that is evaluated to determine whether a job can execute. A job can execute only when the value of the release condition is nonzero or true. You supply the release condition when you define the job. The default release condition is one, which is always true.

Run

Current run number in cases where there are multiple runs. The run number can be accessed through the expression `Model.RunNumber`.

Seed

Random number seed used to run a model. The random number seed generates a set of random numbers that are used to calculate task execution times and probabilistic paths taken through the network. The random number seed can be set in the Execution dialog box. For multiple runs of a model, `Model.RandomSeed` contains the random seed used to generate the particular run.

Snapshot

An option that records the values of specified variables at particular points during model execution—for example, when a task starts or ends, when an entity enters or leaves a queue, at the end of a run, or at specific clock times. When you run a model with snapshots turned On, the values are recorded in a results document that you can use to generate graphs and statistics.

Spinner Task

A task with a path that goes back to itself. Spinner tasks are used to generate multiple entities to travel through the network.

Subnetwork

A network that is inside another network. In a model, all networks are subnetworks except for the top network (level 1). You can create a subnetwork by clicking on your network diagram with the Network tool, then selecting the subnetwork with the pointer and opening its diagram. You draw the subnetwork diagram in the same way as you draw the main network diagram.

Task

The basic building blocks of a model. A task is defined by timing information, execution constraints (or release conditions), effects of the task on the system, and routing information concerning following tasks and subnetworks. Each time a task executes, the execution time is randomly calculated within the time distribution parameters you supply. You can create a task with the Task tool and define it by opening and filling in a Task Description dialog box.

Variable

A quantity with a fixed name that can vary in value. When a variable name is encountered in an expression, it substitutes the current value of the variable. Variables are useful for representing changeable aspects of the system you are simulating. You can change the value of a variable from any expression in your model—for example, as a task or queue effect or in a scenario event.

Index

Symbols

.mod files 62
.res files 64
.xml files 64

A

Abort function 255
Abort(array) function 255
Abs function 266
Acos function 266
adding
    charts 184
    comments to the Animator 225
    constraints 322
    custom toolbar buttons 54
    decision variables 320
    default toolbar buttons 53
    functions 133
    images to Animator display 219
    networks 91
    new toolbars 50
    objectives 325
    objects to Animator display 216
    objects to the network diagram 71
    scenario events 141
    snapshots 146
    task network comments 152
    tasks 100
    variables 116
    watches 129
    way points 223
adjustment operator 239
aligning objects on network diagram . . . 73
angle in Animator functions 283
Animator
    adding charts 222
    adding comments 225
    adding images to palette 218
    adding objects 216
    adding way points 223
    angle 283
    clearing background 221
    CommentID 282
    copying display 228
    deleting charts 222
    deleting images from palette 218
    deleting way points 224
    developing 227
    displaying 216
    Duration 282
    editing comments 225
    enabling edit mode 216
    EndID 282
    executing 228
    functions summary 226
    height 283
    IEnumerable 281
    Image ID 281
    Image IDs 281
    menu 215
    ObjectID 281
    overview of window 212
    palette 213
    PointList 282
    printing display 229
    properties window 214
    Rft 283
    setting background 220
    StartID 282
    supported formats 218
    text 283
    tree view 213
    TypeID 281
    using functions 226
    WayPointID 281
    WayPointIDList 283

width 283
X coordinate 282
Y coordinate 282
Animator comments
adding 225
displaying descriptions 225
editing in the Animator 225
overview 224
Animator functions 281
angle parameter 283
ClearBackground 284
ColorFilter 284
CommentID parameter 282
CopyImage 285
CreateComment 285
CreateImage 286
CreateImageAtWayPoint 287
CreateWayPoint 287
Duration parameter 282
EndID parameter 282
FollowPath 288
GetImage 288
GetWayPoint 288
height parameter 283
HideComment 289
HideComments 289
HideImage 289
HideImages 289
ImageID parameter 281
MoveComment 290
MoveImage 290
MoveImageDown 290
MoveImageToBottom 291
MoveImageToPoints 291
MoveImageToTop 292
MoveImageToWayPoints 292
MoveImageUp 291
ObjectID parameter 281
PlaceImage 292
PointList parameter 282
RemoveComment 293
RemoveComments 293
RemoveImage 293
RemoveImages 293
rft parameter 283
RotateFlipImage 294
RotateImage 294
ScaleImage 294
SetBackground 295
SetBottomImageText 295
SetCommentText 295
SetTopImageText 295
ShowComment 296
ShowComments 296
ShowImage 296
ShowImages 297
StartID parameter 282
text parameter 283
TypeID parameter 281
WaypointID parameter 281
WayPointIDList parameter 283
width parameter 283
X parameter 282
Y parameter 282
appearance
group icon 98
network icon 93
task icon 112
Appearance and Notes tab
comments 155
tasks 112
arbitrary variables 119
area charts 189
arithmetic operators 237
addition 238
decrement 238
division 237
grouped operations 237
increment 238
multiplication 237
remainder division 238
subtraction 238
arrays
definition 119

dimension 120
example 119
Asin function 266
assignment operator 239
Atan function 267
Atan2 function 267
auto layout for network diagram 74
auto-hide mode
disabling 46
displaying windows 46
windows 45

B

background
clearing from Animator display 221
color of toolbar 57
setting for Animator display 220
bar charts 190
Beep function 256
beginning effect
color in event queue window 168
definition 103
return statement 103
bernoulli distribution 302
Bernoulli function 273
beta distribution 302
Beta function 274
BigMul function 267
binomial distribution 303
Binomial function 274
boolean variables 118
bubble charts 190
buttons on toolbar
adding default 53
customizing 51
removing default 53

C

C# syntax 231
cascading If statements 244
case-sensitive names
functions 133
variables 116
Ceiling function 267
changing
active window 47
levels in network diagram 79
charts
adding to Animator display 222
adding to simulation 184
area 189
bar 190
bubble 190
changing values in data editor 206
contour 191
copying 209
cube 191
curve 192
curve area 192
data editor 206
deleting 188
deleting from Animator display 222
disabling update during execution . 170
displaying 184, 186
displaying data editor 206
displaying defined 183
displaying descriptions 184
doughnut 193
editing 188
enabling update during execution . 170
financial 193
formatting 199
formatting label text 207
formatting legend box 208
gantt 194
General tab 200
graph series 187
hiding data editor 206
line 194
parameters 185
pareto 195

pasting 209
pie 195
printing 210
Properties tab 185
pyramid 196
radar 196
saving as image file 210
scatter 197
Series List tab 186
Series tab 202
step 197
surface 198
toolbar 199
types 189
Y Axis tab 204
checking
logic errors 173
syntax errors 172
ClearBackground function 284
clearing output window 167
Clock variable 121
closing windows 47
color
event queue window 168
group icon 98
network icon 93
task icon 112
ColorFilter function 284
commas in expressions 235
comment variables
disabling update display 170
displaying updates 170
CommentID in Animator functions 282
comments
disabling during execution 171
See Animator comments
See task network comments
comments in expressions 235
Compiler tab, execution settings 164
constants
defined 232
functions 266
constraint feasible, definition 345
constraints
adding 322
definition 317
deleting 324
displaying 322
editing 323
feasibility 324
using 324
continuous probability distributions . . . 300
contour charts 191
CopyImage function 285
copying
Animator display 228
charts 209
graphical network diagram 81
objects 81
text 81
Cos function 268
Cosh function 268
CreateComment function 285
CreateEntity function 256
CreateImage function 286
CreateImageAtWayPoint function 287
CreateWayPoint function 287
creating
groups 95, 96
models 60
network diagram 70
cube charts 191
curve area charts 192
curve charts 192
customizing toolbar buttons 51
cutting
objects 81
text 81

D

data
queue data files 181

snapshot data file 182
task data files 180
trace data files 180
data editor
displaying 206
hiding 206
overview 206
Data tab, execution settings 160
decision code 108
decision nodes
decision code 108
default 107
types 107
decision types
multiple 107
probabilistic 107
tactical 108
decision variables
adding 320
definition 317
deleting 322
displaying 320
editing 321
decrement operator 238
default decision nodes 107
default window configuration 30
deleting
charts 188
constraints 324
decision variables 322
entity attributes 127
functions 138
networks 94
objectives 327
objects 82
objects on network diagram 72
paths on network diagram 76
scenario events 144
snapshots 151
task network comments 156
text 82
variables 121
watches 130
way points 224
departing effect for queues 111
design variable, defined 346
developing Animator display 227
dimensions for arrays 120
Direction function 256
disabling
auto-hide mode 46
network animation 172
update of charts 170
update of comments 171
discrete probability distributions 300
discrete uniform distribution 303
discrete variable, defined 346
DiscreteUniform function 274
displaying
Animator window 216
chart descriptions 184
charts 186
comment descriptions in Animator . 225
constraints 322
decision variables 320
defined charts 183
defined entity attributes 123
defined functions 132
defined scenario events 140
defined snapshots 145
defined variables 115
entity attribute descriptions 123
execution settings 158
function descriptions 132
hidden windows 44
logic for paths 76
network animation 171
network descriptions 90
network properties in HTML 85
object properties in HTML 83
objectives 325
scenario event descriptions 140
snapshot descriptions 145
task descriptions 99

task network comment descriptions 152
toolbars 49
updates to comment variables 170
variable descriptions 115
variables on task network diagram .152
windows in auto-hide mode 46
Distance function 257
distribution functions
Bernoulli 273
Beta 274
Binomial 274
DiscreteUniform 274
Exponential 275
ExtremeValueTypeA 275
ExtremeValueTypeB 275
Gamma 276
Geometric 276
InverseGaussian 276
Logistic 277
LogLogistic 277
Lognormal 277
NegativeBinomial 278
Normal 278
overview 272
Pareto 278
PearsonType5 279
PearsonType6 279
Poisson 279
Rectangular 280
Triangular 280
Weibull 280
Distributions variable 121
DivRem function 268
do statements 247
dockable windows 41
docking windows 42
Document windows
defined 40
tab groups 42
doubles variables 118
doughnut charts 193
drawing paths between tasks 75
duration attribute 122
duration in Animator functions 282

E

edit mode, Animator 216
editing
charts 188
comments in the Animator 225
constraints 323
decision variables 321
entity attributes 126
execution settings 159
functions 138
networks 94
objectives 327
scenario events 144
snapshots 150
task network comments 156
variables 120
watches 130
enabling
network animation 171
queue data collection in task 110
queue information collection 162
snapshots 163
task data collection 101
task information collection 162
EndID in Animator functions 282
ending effect 103
color in event queue window 168
entering effect 111
entities
multiple 113
split 113
entity attributes
deleting 127
displaying defined 123
displaying descriptions 123
duration 122
editing 126

group 128
ID 128
parameters 125
tag 128
time 128
type 128
uniqueID 128
Entity variable 121
entity variable type 118
errors
logic 173
syntax 172
event queue window
colors 168
overview 35
viewing scheduled events 168
events. *See* scenario events
executing Animator display 228
execution settings
Compiler tab 164
Data tab 160
displaying 158
editing 159
enable layout change 163
enable snapshots 163
enable task information collection . 162
extra namespaces 164
extra references 164
initialization code 164
line by line debugging 161
number of runs 160
output type 161
parameters 159
queue information collection 162
random seed 161
realtime mode 161
speed factor 163
Start ID 161
execution speed 177
Exp function 268
exponential distribution 304
Exponential function 275
exporting
.res 64
.xml 64
location of files 64
model information 63
queue information 63
snapshot data 63
snapshots automatically 148
task data 63
type of files 64
expressions
adjustment operators 239
arithmetic operators 237
assignment operator 239
C# syntax 231
cascading If statement 244
commas 235
comments 235
constants 232
do statements 247
for loops 246
foreach statements 249
functions 234
if-then else statements 242
logical operators 240
return statements 236
switch statements 247
variables 233
while statements 245
Expressions tab, snapshots 149
extra namespaces 164
extra references 164
Extreme Value Type A distribution 304
Extreme Value Type B distribution 305
ExtremeValueTypeA function 275
ExtremeValueTypeB function 275

F

FIFO queue type 110
financial charts 193

floating variables 118
floating windows 40
Floor function 269
flyover text 33
FollowPath function 288
font style for toolbar 57
for loops 246
foreach statements 249
foreground color of toolbar 57
formatting
  charts 199
  label text on charts 207
  legend box on charts 208
functions
  Abort 255
  Abort(array) 255
  Abs 266
  Acos 266
  adding 133
  Asin 266
  Atan 267
  Atan2 267
  Beep 256
  Bernoulli 273
  Beta 274
  BigMul 267
  Binomial 274
  Ceiling 267
  ClearBackground 284
  code 134
  ColorFilter 284
  constants 266
  CopyImage 285
  Cos 268
  Cosh 268
  CreateComment 285
  CreateEntity 256
  CreateImage 286
  CreateImageAtWayPoint 287
  CreateWayPoint 287
  deleting 138
  Direction 256
  DiscreteUniform 274
  displaying defined 132
  displaying descriptions 132
  Distance 257
  distribution 272
  DivRem 268
  editing 138
  Exp 268
  Exponential 275
  ExtremeValueTypeA 275
  Floor 269
  FollowPath 288
  Gamma 276
  Geometric 276
  GetImage 288
  GetWayPoint 288
  Halt 257
  HideComment 289
  HideComments 289
  HideImage 289
  HideImages 289
  IEEERemainder 269
  in expressions 234
  InverseGaussian 276
  Kill 257
  Log 269
  Log10 269
  Logistic 277
  LogLogistic 277
  Lognormal 277
  math 265
  Max 270
  Message 258
  Min 270
  model 253
  MoveComment 290
  MoveImage 290
  MoveImageDown 290
  MoveImageToBottom 291
  MoveImageToPoints 291
  MoveImageToTop 292
  MoveImageToWayPoints 292

MoveImageUp 291
name restrictions 133
NegativeBinomial 278
Normal 278
overview 131
Parameters tab 137
Pareto 278
Pause 258
PearsonType5 279
PearsonType6 279
PlaceImage 292
Poisson 279
Pow 270
PrintOutput 259
Properties tab 134
Random 259
RandomInteger 259
RandomSeed 260
Rectangular 280
RemoveComment 293
RemoveComments 293
RemoveImage 293
RemoveImages 293
Resume 260
Resume (array) 261
Return Information tab 135
RotateFlipImage 294
RotateImage 294
Round 270
RunNumber 261
ScaleImage 294
SendExternalVariable 261
SetBackground 295
SetBottomImageText 295
SetCommentText 295
SetTopImageText 295
ShowComment 296
ShowComments 296
ShowImage 296
ShowImages 297
Sign 271
Sin 271
Sinh 271
Sqrt 271
Start 262
Stop 262
Stop(array) 263
Suspend 263
Suspend (array) 264
syntax overview 252
Tan 272
Tanh 272
ToString 264
Triangular 280
TriggerSnapshot 264
type 136
usage 252
WallClock 265
Weibull 280

G

gamma distribution 305
Gamma function 276
gantt charts 194
General tab, charts 200
geometric distribution 306
Geometric function 276
GetImage function 288
GetWayPoint function 288
global variables 233
graph series 187
graphical network diagram, copying 81
group attribute 128
groups
    changing icon appearance 98
    creating 95, 96
    overview 95
    properties 97
    style of icon 98
    ungrouping 99

H

Halt function 257
hardware key 13
height in Animator 283
heuristic, defined 347
HideComment function 289
HideComments function 289
HideImage function 289
HideImages function 289
hiding windows 44

I

ID attribute 128
IEEERemainder function 269
IEnumerable 281
if-then else statements 242
ImageID in Animator functions 281
ImageIDs in Animator functions 281
ImageIDs parameter 281
images in Animator
    adding to display 219
    adding to palette 218
    deleting from palette 218
    resizing 220
    supported formats 218
increment operator 238
initial value for variables 119
initialization code 164
integer variables 118
inverse gaussian distribution 306
InverseGaussian function 276

J

jumping to objects 78

K

keywords, reserved 116
Kill function 257

L

launch effect 103
legend box on charts, formatting 208
LIFO queue type 110
line by line debugging 161
line charts 194
line debugger window, overview . . 38, 174
local variables 233
Log function 269
Log10 function 269
logic errors 173
logical operator 240
logistic distribution 307
Logistic function 277
loglogistic distribution 307
LogLogistic function 277
lognormal distribution 308
Lognormal function 277

M

Main tab, tasks 102
math functions
    Abs 266
    Acos 266
    Asin 266
    Atan 267
    Atan2 267
    BigMul 267
    Ceiling 267
    constants 266
    Cos 268
    Cosh 268
    DivRem 268
    Exp 268
    Floor 269
    IEEERemainder 269
    Log 269

Log10 269
Max 270
Min 270
overview 265
Pow 270
Round 270
Sign 271
Sin 271
Sinh 271
Sqrt 271
Tan 272
Tanh 272
Max function 270
mean time 105
menu bar 32
Message function 258
metaheuristic, defined 347
metaheuristics 340
Min function 270
model execution
pausing 175
resuming 175
setting speed 177
starting 175
stepping through 176
stopping 176
model functions
Abort 255
Abort(array) 255
Beep 256
CreateEntity 256
Direction 256
Distance 257
Halt 257
Kill 257
Message 258
overview 253
Pause 258
PrintOutput 259
Random 259
RandomInteger 259
RandomSeed 260
Resume 260
Resume(array) 261
RunNumber 261
SendExternalVariable 261
Start 262
Stop 262
Stop(array) 263
Suspend 263
Suspend (array) 264
ToString 264
TriggerSnapshot 264
WallClock 265
Model variable 122
models
creating 60
exporting information 63
opening existing 61
opening from previous versions 62
saving 62
MoveComment function 290
MoveImage function 290
MoveImageDown function 290
MoveImageToBottom function 291
MoveImageToPoints function 291
MoveImageToTop function 292
MoveImageToWayPoints function 292
MoveImageUp function 291
moving
objects on network diagram 73
toolbar sections 48
windows 46
multiple decision type 107
multiple entities 113

N

naming restrictions
functions 133
scenario events 141
snapshots 146
variables 116

watches 129
negative binomial distribution 309
NegativeBinomial function 278
network animation
disabling 172
enabling 171
network comments
color 155
style 155
network diagram
adding objects 71
aligning objects 73
changing levels 79
comments 151
creating 70
deleting objects 72
designating start node 77
displaying variables 152
drawing paths 75
events 139
functions 131
groups 95
jumping to objects 78
moving objects 73
overview of window 34
panning 78
printing 86
removing paths 76
snap to grid 72
snapshots 145
tasks 99
using auto layout 74
variables 114
viewing object properties in HTML . . 83
viewing properties in HTML 85
Watches 129
zooming in 77
zooming out 77
networks
adding 91
changing icon appearance 93
deleting 94
displaying descriptions 90
editing 94
overview 90
parameters 92
style of icon 93
normal distribution 309
Normal function 278
Notes tab
scenario events 143
snapshots 150
number of runs 160

O

object variables 118
ObjectID in Animator functions 281
objectives
adding 325
definition 317
deleting 327
displaying 325
editing 327
objects
copying 81
cutting 81
deleting 82
displaying properties in HTML 83
pasting 82
opening
existing models 61
models from previous versions 62
operators
adjustment 239
arithmetic 237
assignment 239
logical 240
optimization, defined 348
options for OptQuest 327
OptQuest
adding constraints 322
adding decision variables 320

adding objectives 325
checking for errors 329
classical methods 339
constraints 322
decision variables 317, 319
deleting constraints 324
deleting decision variables 322
deleting objectives 327
displaying constraints 322
displaying decision variables 320
displaying objectives 325
editing constraints 323
editing decision variables 321
editing objectives 327
elements 317
factors affecting performance 332
glossary 344
halting 329
interaction with Micro Saint Sharp . 316
interpreting results 330
metaheuristics 340
methodology 339
objective 317
objectives 325
options 327
overview 317
preparing the model 318
running optimization 329
scatter search 343
starting 319
suggestions for using 331
tabu search 343
output constraint, defined 348
output type 161
output window
clearing 167
information 180
overview 37
saving results 167
saving results to clipboard 168
viewing simulation 166

P

palette
Animator 213
task network 37
panning the network diagram 78
Parameters tab, functions 137
pareto charts 195
pareto distribution 310
Pareto function 278
pasting
charts 209
objects 82
text 82
paths
deleting 76
displaying logic 76
drawing between tasks 75
removing 76
Paths tab 106
Pause function 258
pausing model execution 175
Pearson Type 5 distribution 310
Pearson Type 6 distribution 311
PearsonType5 function 279
PearsonType6 function 279
pie charts 195
PlaceImage function 292
PointList in Animator functions 282
poisson distribution 311
Poisson function 279
Pow function 270
printing
Animator display 229
charts 210
network diagram 86
PrintOutput function 259
priority, queues 111
probabilistic decision type 107
probability distributions
bernoulli 302
beta 302

binomial 303
continuous 300
discrete 300
discrete uniform 303
exponential 304
Extreme Value Type A 304
Extreme Value Type B 305
gamma 305
geometric 306
inverse gaussian 306
logistic 307
loglogistic 307
lognormal 308
negative binomial 309
normal 309
overview 300
pareto 310
Pearson Type 5 310
Pearson type 6 311
poisson 311
rectangular (uniform) 312
references 302
scale parameter 301
shape parameter 301
triangular 312
weibull 313
Properties tab
charts 185
functions 134
scenario events 142
snapshots 147
properties window
expanding 35
overview 35
sorting 35
push pin icon 45
pyramid charts 196

Q

queue data collection
enabling in execution settings 162
enabling in task 110
queue data files
exporting 63
information in 181
Queue tab 109
queue types
FIFO 110
LIFO 110
sorted 110
queues
departing effect 111
entering effect 111
priority 111
types 110

R

radar charts 196
Random model function 259
random seed 161
RandomInteger function 259
RandomSeed function 260
real variables 118
realtime mode 161
rectangular distribution 312
Rectangular function 280
redisplaying hidden windows 44
release condition 102
remainder division 238
RemoveComments function 293
RemoveImage function 293
RemoveImages function 293
removing
custom toolbar buttons 56
default toolbar buttons 53
paths on network diagram 76
toolbars 49
repeating interval 142
repeating snapshots 148
replacing text 66

reserved names 116
resetting
default window configuration 30
toolbar configuration 56
resizing images in Animator displays . . 220
resizing windows 47
Resume function 260
Resume(array) function 261
resuming model execution 175
Return Information tab, functions 135
return statements 236
Rft in Animator functions 283
RotateFlipImage function 294
RotateImage function 294
Round function 270
running optimization 329
RunNumber function 261

S

saving
charts as image file 210
models 62
output window to clipboard 168
output window to file 167
trace data files 167
trace data files to clipboard 168
ScaleImage function 294
scatter charts 197
scatter search 343
scenario events
adding 141
code 143
color in event queue window 168
deleting 144
disabling update during execution . 169
displaying defined 140
displaying descriptions 140
editing 144
naming restrictions 141
Notes tab 143
overview 139
parameters 141
Properties tab 142
repeating interval 142
scoping, variables 233
searching for text 65
Sections window, overview 34
selecting text or items 80
SendExternalVariable function 261
sequence variable, defined 349
series in charts 187
Series List tab, charts 186
Series tab, charts 202
SetBackground function 295
SetBottomImageText function 295
SetCommentText function 295
SetTopImageText function 295
ShowComments function 296
ShowImage function 296
ShowImages function 297
Sign function 271
Sin function 271
Sinh function 271
snap to grid, network diagram 72
snapshots
adding 146
auto export 148
color in event queue window 168
deleting 151
displaying defined 145
displaying descriptions 145
editing 150
enabling 163
exporting 63
Expressions tab 149
names 146
Notes tab 150
overview 145
parameters 146
Properties tab 147
repeating 148
trigger ID 148

trigger type 147
sorted queue type 110
sorting items in properties window 35
speed factor 163
speed, execution 177
spinner tasks 113
split entities 113
Sqrt function 271
standard deviation 105
Start function 262
Start ID, execution settings 161
start node, network diagram 77
StartID in Animator functions 282
starting model execution 175
starting OptQuest 319
status bar 39
step charts 197
stepping through model execution 176
Stop (array) function 263
Stop function 262
stopping model execution 176
string variables 118
style
groups 98
network comments 155
network icon 93
task icon 112
surface charts 198
Suspend (array) function 264
Suspend function 263
switch statements 247
syntax errors 172
system variables 121
Animator 122
Clock 121
Distributions 121
Entity 121
Model 122

T

tab groups
displaying 42
Document windows 42
tabu search 343
tactical decision type 108
tag attribute 128
Tan function 272
Tanh function 272
task data collection
enabling in execution settings 162
enabling in tasks 101
task data files
exporting 63
information 180
task network comments
adding 152
Appearance and Notes tab 155
color 155
deleting 156
displaying descriptions 152
editing 156
overview 151
parameters 153
style 155
updating during execution 170
Variables to Display tab 154
task network diagram
See network diagram
task network palette
overview 37
tools 71
tasks
adding 100
Appearance and Notes tab 112
beginning effect 103
changing icon appearance 112
departing effect. 111
displaying descriptions 99
ending effect 103
entering effect for queue 111

launch effect 103
Main tab 102
mean time 105
overview 99
parameters 100
Paths tab 106
priority 111
Queue tab 109
release condition 102
spinner 113
standard deviation 105
style of icon 112
time distribution 105
Timing tab 104
text
copying 81
cutting 81
deleting 82
pasting 82
replacing 66
searching 65
text in Animator functions 283
time distribution for tasks 105
time entity attribute 128
Timing tab 104
Tool windows
defined 40
tab groups 42
toolbar buttons
adding custom 54
adding default 53
removing custom 56
toolbars
adding custom buttons 54
adding new 50
background color 57
charts 199
customizing 47
customizing buttons 51
displaying 49
flyover text 33
font setting 57
foreground color 57
look and feel 57
moving sections 48
overview 33
removing 49
removing custom buttons 56
removing default buttons 53
resetting configuration 56
ToString function 264
trace data files
clearing output 167
information in 180
saving results 167
saving to clipboard 168
viewing 166
tree view window
expanding items 36
overview 36
triangular distribution 312
Triangular function 280
trigger ID 148
trigger type 147
TriggerSnapshot function 264
type attribute 128
TypeID in Animator functions 281

U

uniform distribution 312
uniqueID attribute 128

V

variables
adding 116
arbitrary 119
array dimensions 120
array setting 119
boolean 118
decision 317
defined 233

deleting 121
displaying defined 115
displaying descriptions 115
doubles 118
editing 120
entity 118
floating 118
global 233
initial value 119
integer 118
local 233
name restrictions 116
object 118
parameters 117
real 118
scoping 233
string 118
system 121
types 118

W

WallClock function 265
watches
adding 129
deleting 130
editing 130
names 129
watches window, overview 38
way points
adding 223
defined 223
deleting 224
WayPointID in Animator functions 281
WayPointIDList in Animator functions . . 283
weibull distribution 313
Weibull function 280
while statements 245
width in Animator functions 283
window components
event queue 35
line debugger window 38, 174
output window 37
overview 31
properties window 35
Sections window 34
status bar 39
task network 34
task network palette 37
title bar 32
toolbar 33
tree view 36
watches window 38
windows
Animator 212
auto-hide mode 45
changing active 47
closing 47
disabling auto-hide mode 46
displaying in auto-hide mode 46
docking 42
Document 40
hiding 44
line debugger 38, 174
making dockable 41
making floating 40
manipulating 40
moving 46
redisplaying 44
resizing 47
Tool 40

X

X coordinate, Animator functions 282

Y

Y Axis tab, charts 204
Y coordinate, Animator functions 282

Z

zooming the network diagram 77

■ Index